Applicant : David S. Taubman
Serial No. : 10/631,884
Filed : July 29, 2003

Attorney's Docket No.: 10991918-2

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant : David S. Taubman
Serial No. : 10/631,884
Filed : July 29, 2003
Title : IMPROVED BLOCK ENTROPY CODING IN EMBEDDED BLOCK CODING WITH OPTIMIZED TRUNCATION IMAGE COMPRESSION

Art Unit : 2625
Examiner : Chen, Wenpeng

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

DECLARATION UNDER 37 CFR § 1.131

I, David S. Taubman, hereby declare as follows.

1. I am the sole inventor of the subject matter recited in the pending claims of the above-identified patent application.

2. Prior to December 17, 1999, I completed my invention as described and claimed in the subject application in this country, as evidenced by the following.

a. Prior to December 17, 1999, I conceived the idea of an image data compression system that included a decomposition processor coupled to an arithmetic coder. In operation, the decomposition processor would decompose the image data into code-blocks of coefficients using a transform, where each code-block included a plurality of bit-planes from a most significant bit-plane to a least significant bit-plane. The arithmetic coder would form an encoded bit-stream by coding bit-planes of coefficient data in the code-blocks according to an arithmetic coding scheme. The arithmetic coder also would be constructed such

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service as First Class Mail in an envelope addressed to: Commission for Patents, PO Box 1450, Alexandria, VA 22313-1450 on:

March 14, 2006

Date

(Signature of person mailing papers)

Edouard Garcia

(Typed or printed name of person mailing papers)

that coefficient data from at least one bit-plane is not subjected to said arithmetic coding scheme so as to be included in the encoded bit-stream without arithmetic coding.

b. Prior to December 17, 1999, I made a physical embodiment of the image data compression system of ¶ 2.a in the form of a computer program, and I operated this physical embodiment to carry out the steps of the associated method in a manner demonstrative of the workability of the idea of ¶ 2.a.

c. The reduction to practice of the physical embodiment of ¶ 2.b is evidenced by the invention disclosure document entitled "Various improvements to the block entropy coder in the EBCOT image compression algorithm" that is attached hereto as Exhibit A.

d. Page 3 of Exhibit A describes an image data compression approach in accordance with which "all of the binary symbols generated in the all of the binary symbols generated in the 'significance propagation' and 'magnitude refinement' coding passes (see Section II-1.2 in the Attachment) representing bits in bit-planes $p < p_0 - K$ are written directly into the bit-stream as raw binary digits, entirely bypassing the arithmetic coder, where p_0 denotes the most significant bit-plane in which any sample in the relevant code-block becomes significant and K is a parameter which we suggest should be set to $K = 3$."

e. Page 3 of Exhibit A also describes the following modification that was made to the MQ coder in the physical embodiment of ¶ 2.b:

> It should be pointed out that an additional modification to the Elias termination procedure was required in order to ensure that this "lazy" mode could be used in conjunction with the MQ coder. Specifically, since the MQ coder is byte-oriented, with a bit-stuffing rather than carry propagation policy for dealing with carry generation at the encoder, the arbitrary bit-stream suffices which can be generated by the emission of raw uncoded bits can generate illegal bit-stream for a previous MQ-coded pass. To avoid this

difficulty, we modified the Elias termination implementation to allow for truly arbitrary suffices; the details of this modification are not warranted by the scope of this document, but they are adequately described by comments in the source code.

f. The results of tests on the operation of the physical embodiment of ¶2.b are presented in Tables 1 and 2 on pages 5 and 6 of Exhibit A, respectively, and are summarized in the bulleted paragraphs at the end of page 6 of Exhibit A. These results demonstrated that, relative to the other compression approaches tested, the physical embodiment of ¶2.b substantially reduced the number of symbols which must be arithmetically coded at high bit-rates while maintaining comparable decompressed image quality for any given bit-rate.

3. I declare that all statements made herein of my own knowledge are true and that all statements made on declaration and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the application or any patent issuing thereon.

Respectfully submitted,

Date: 6 March, 2006

David S. Taubman

Applicant : David S. Taubman
Serial No. : 10/631,884
Filed : July 29, 2003

Attorney's Docket No.: 10991918-2

EXHIBIT A

H **INVENTION DISCLOSURE** PAGE ONE OF 2 CPL

PDNO 10991918 DATE RCVD 10-28-99 ATTORNEY TRC

Instructions: *The information contained in this document is* ***COMPANY CONFIDENTIAL*** *and may not be disclosed to others without prior authorization. Submit this disclosure to the HP Legal Department as soon as possible. No patent protection is possible until a patent application is authorized, prepared, and submitted to the Government.*

Descriptive Title of Invention:

"Various improvements to the block entropy coder in the EBCOT image compression algorithm"

Name of Project:

Product Name or Number:

Was a description of the invention published, or are you planning to publish? If so, the date(s) and publication(s):
Planning to distribute to JPEG2000 committee by 30 June 1999.

Was a product including the invention announced, offered for sale, sold, or is such activity proposed? If so, the date(s) and location(s):
No

Was the invention disclosed to anyone outside of HP, or will such disclosure occur? If so, the date(s) and name(s):
Yes, various selected members of the JPEG2000 committee (University of Arizona, SAIC, Canon Research France, CISRA Australia, Sharp Labs USA. This limited disclosure was made on March 18, 1999 in Seoul, Korea.

If any of the above situations will occur within 3 months, call your IP attorney or the Legal Department now at 1-553-3061 or 408-553-3061.

Was the invention described in a lab book or other record? If so, please identify (lab book #, etc.)

Was the invention built or tested? If so, the date:

Yes, on June 1, 1999.

Was this invention made under a government contract? If so, the agency and contract number:

No

Description of Invention: *Please preserve all records of the invention and attach additional pages for the following. Each additional page should be signed and dated by the inventor(s) and witness(es).*

A. Prior solutions and their disadvantages (if available, attach copies of product literature, technical articles, patents, etc.).
B. Problems solved by the invention.
C. Advantages of the invention over what has been done before.
D. Description of the construction and operation of the invention (include appropriate schematic, block, & timing diagrams; drawings; samples; graphs; flowcharts; computer listings; test results; etc.)

Signature of Inventor(s): Pursuant to my (our) employment agreement, I (we) submit this disclosure on this date: [11 JUNE, 1999].

Employee No.	Name	Signature	Telnet	Mailstop	Entity & Lab Name
46142	DAVID TAUBMAN			C919, CRL	

(If more than four inventors, include additional information on another copy of this form and attach to this document)

Signature of Witness(es): *(Please try to obtain the signature of the person(s) to whom invention was first disclosed.)*

The invention was first explained to, and understood by, me (us) on this date: [MARCH 17, 1999]

Full Name	Signature	Date of Signature
ALEXANDER DRUKAREV	Alexander Drukarev	6/25/99
Full Name	Signature	Date of Signature
CHRISTOS CHRYSAFIS	Christos Chrysafis	6/25/99

Inventor & Home Address Information: *(If more than four inventors, include addl. information on a copy of this form & attach to this document)*

Inventor's Full Name

David Scott Taubman

Street

20 Wingate Ave.

City | State | Zip

Eastwood, NSW, 2073 Australia

Do you have a Residential P.O. Address? P.O. BOX | City | State | Zip

Greeted as *(nickname, middle name, etc.)* | Citizenship

David | **Australia**

Inventor's Full Name

Street

City | State | Zip

Do you have a Residential P.O. Address? P.O. BOX | City | State | Zip

Greeted as *(nickname, middle name, etc.)* | Citizenship

Inventor's Full Name

Street

City | State | Zip

Do you have a Residential P.O. Address? P.O. BOX | City | State | Zip

Greeted as *(nickname, middle name, etc.)* | Citizenship

Inventor's Full Name

Street

City | State | Zip

Do you have a Residential P.O. Address? P.O. BOX | City | State | Zip

Greeted as *(nickname, middle name, etc.)* | Citizenship

Various improvements to the block entropy coder in the EBCOT image compression algorithm.

David Taubman

Patent disclosure

This invention addresses some modifications to the EBCOT image compression method (see invention disclosure "Image compression system based on EBCOT"). The modifications are described in the context of the system, which uses sub-blocks in the EBCOT coding engine. The detailed description of the EBCOT system with sub-blocks is given in the document "Reduced complexity entropy coding with sub-blocks", shown as an attachment to this disclosure. This document "Reduced complexity entropy coding with sub-blocks" is a draft of the submission to the ISO SC29WG01 committee working on the new image compression standard JPEG 2000. All the references to various sections in this Invention Disclosure refer to the sections on the attached JPEG 2000 document.

This invention proposes a method, which both reduces the average number of arithmetically coded symbols and also reduces the maximum number of coding passes in which symbols might need to be arithmetically coded, which can be of advantage in simplifying hardware implementations. The idea is very simple and is connected with techniques used in Ricoh's "CREW" algorithm. Specifically, we begin by observing that the probability models for many coding contexts attain distributions close to uniform in the least significant bit-planes. It is a waste of effort to use the arithmetic coding engine to code these binary symbols; instead, we prefer to send them as raw binary digits. Although it is difficult to interleave raw binary digits into an arithmetically coded bit-stream, it is possible to bypass the arithmetic coding engine altogether for an entire sub-bitplane coding pass provided the arithmetic coder is terminated at the end of the previous coding pass (using the Elias termination which allows for unique decoding with arbitrary bit-stream suffices) and restarted at the beginning of the next pass which requires arithmetic coding. This is a subset of the behaviour offered by the "-Crestart" option and for our experiments (see Attachment) so the software implementation supports the behaviour both with and without the costly reinitialization of coding context states which goes with the "-Crestart" mode. For our experiments, however, we prefer to use this "lazy" mode only in conjunction with "-Crestart" and sub-block causal contexts for reasons which will be explained shortly. To be specific, in the proposed, optional modification, all of the binary symbols generated in the "significance propagation" and "magnitude refinement" coding passes (see Section II-1.2 in the Attachement) representing bits in bit-planes $p < p_0 - K$ are written directly into the bit-stream as raw binary digits, entirely bypassing the arithmetic coder, where p_0 denotes the most significant bit-plane in which any sample in the relevant code-block becomes significant and K is a parameter which we suggest should be set to $K = 3$. It should be pointed out that an additional modification to the Elias termination procedure was required in order to ensure that this "lazy" mode could be used in conjunction with the MQ coder. Specifically, since the MQ coder is byte-oriented, with a bit-stuffing rather than carry propagation policy for dealing with carry generation at the encoder, the arbitrary bit-stream suffices which can be generated by the emission of raw uncoded bits can generate illegal bit-stream for a previous MQ-coded pass. To avoid this difficulty, we modified the Elias termination implementation to allow for truly arbitrary suffices; the details of this modification are not warranted by the scope of this document, but they are adequately described by comments in the source code.

One advantage of the modification is that it substantially reduces the number of symbols which must be arithmetically coded at high bit-rates. Also, since we usually encode all code-blocks in an image at a high rate before truncating down to a final target bit-rate, this scheme substantially reduces the number of symbols which must typically be encoded and hence reduces the encoding time. We find, for example, that CPU times for reversible compression are typically reduced by 30%. On the other hand, the modification has relatively little effect on compression performance.

The second advantage of the modification is that it substantially reduces the maximum number of coding passes in which arithmetic coding might need to be used. Without the modification, the maximum number of coding passes for any given code-block is $3P_{max} - 2$ where P_{max} is the maximum number of bit-planes

in any given subband and might be on the order of 12 for the lower frequency subbands. On the other hand, with the modification, the maximum number of coding passes for any given code-block is $P_{max} + 2K = P_{max} + 6$. In applications where we intend to use microscopic parallelism to achieve "sample-per-clock" throughput, this means a substantial reduction in the number of parallel arithmetic coding engines which must be included on the chip. Due to the importance of the combination of this mode with parallel encoding/decoding, we provide results in Table 1 for the combination of this so-called "lazy" option with the "-Crestart" option and sub-block causal coding contexts, comparing the performance with the use of sub-block causal coding contexts and "-Crestart" alone.

Table 1 *Comparison of the "lazy coding" option in combination with the parallel options with the parallel options alone, for a sub-block size of 16x16 and a code-block size of 64x64.*

Lenna (512x512)

Par bpp	Par symbols	Par PSNR	Lazy bpp (+/- %)	Lazy Symbols (+/- %)	Lazy PSNR (+/- dB)
0.06226	0.088	28.09	0.06223 (-0.0)	0.085 (-3.6)	28.09
0.12491	0.179	30.97	0.12491	0.167 (-7.0)	30.96 (-0.00)
0.24924	0.362	34.04	0.24970 (+0.2)	0.310 (-14.2)	33.99 (-0.05)
0.49988	0.715	37.21	0.49973 (-0.0)	0.580 (-18.8)	37.14 (-0.07)
0.99878	1.367	40.32	0.99966 (+0.1)	1.041 (-23.9)	40.28 (-0.04)
1.99792	2.500	44.80	1.98401 (-0.7)	1.731 (-30.8)	44.75 (-0.05)

Aerial 2 (2048x2048+A45

Par bpp	Par symbols	Par PSNR	Lazy bpp (+/- %)	Lazy Symbols (+/- %)	Lazy PSNR (+/- dB)
0.06221	0.090	24.60	0.06203 (-0.3)	0.080 (-10.7)	24.59 (-0.00)
0.12477	0.184	26.48	0.12472 (-0.0)	0.164 (-11.2)	26.49 (+0.00)
0.24984	0.345	28.54	0.24991 (+0.0)	0.304 (-12.0)	28.54 (+0.01)
0.49954	0.669	30.60	0.49966 (+0.0)	0.585 (-12.5)	30.60 (+0.00)
0.99905	1.297	33.21	0.99906 (+0.0)	1.089 (-16.0)	33.23 (+0.01)
1.99660	2.430	38.04	1.99556 (-0.1)	2.008 (-17.4)	38.08 (+0.04)

Bike (2048x2560)

Par bpp	Par symbols	Par PSNR	Lazy bpp (+/- %)	Lazy Symbols (+/- %)	Lazy PSNR (+/- dB)
0.06249	0.116	23.75	0.06248 (-0.0)	0.111 (-4.3)	23.74 (-0.01)
0.12475	0.220	26.28	0.12486 (+0.1)	0.204 (-7.3)	26.24 (-0.04)
0.24999	0.421	29.53	0.24987 (-0.0)	0.377 (-10.5)	29.46 (-0.07)
0.49958	0.797	33.40	0.49997 (+0.1)	0.680 (-14.7)	33.28 (-0.12)
0.99840	1.477	37.96	0.99921 (+0.1)	1.166 (-21.1)	37.80 (-0.16)
1.99778	2.706	43.91	1.99976 (+0.1)	1.844 (-31.9)	43.75 (-0.16)

Café (2048x2560)

Par bpp	Par symbols	Par PSNR	Lazy bpp (+/- %)	Lazy Symbols (+/- %)	Lazy PSNR (+/- dB)
0.06249	0.098	19.03	0.06237 (-0.2)	0.096 (-2.9)	19.02 (-0.00)
0.12491	0.205	20.73	0.12490 (-0.0)	0.200 (-2.8)	20.73 (+0.00)
0.24986	0.388	23.07	0.24999 (+0.1)	0.378 (-2.8)	23.07 (+0.00)
0.49984	0.737	26.71	0.49993 (+0.0)	0.694 (-5.9)	26.70 (-0.01)
0.99934	1.417	31.89	0.99976 (+0.0)	1.255 (-11.4)	31.85 (-0.04)
1.99819	2.646	38.92	1.99956 (+0.1)	1.931 (-27.0)	38.85 (-0.07)

Woman (2048x2560)

Par bpp	Par symbols	Par PSNR	Lazy bpp (+/- %)	Lazy Symbols (+/- %)	Lazy PSNR (+/- dB)
0.06233	0.091	25.59	0.06243 (+0.2)	0.087 (-4.0)	25.59 (-0.00)
0.12426	0.184	27.32	0.12459 (+0.3)	0.180 (-2.6)	27.32 (+0.00)
0.24961	0.354	29.94	0.25000 (+0.2)	0.340 (-3.9)	29.93 (-0.00)
0.49971	0.679	33.55	0.49963 (-0.0)	0.627 (-7.6)	33.54 (-0.01)
0.99979	1.311	38.33	0.99989 (+0.0)	1.089 (-17.0)	38.31 (-0.03)
1.99873	2.501	43.94	1.99651 (-0.1)	1.772 (-29.1)	43.91 (-0.02)

Average (Bike, Café, Woman)

Par bpp	Par symbols	Par PSNR	Lazy bpp (+/- %)	Lazy Symbols (+/- %)	Lazy PSNR (+/- dB)
0.06243	0.102	22.79	0.06243 (-0.0)	0.098 (-3.8)	22.78 (-0.01)
0.12464	0.203	24.78	0.12478 (+0.1)	0.194 (-4.3)	24.76 (-0.01)
0.24982	0.388	27.51	0.24995 (+0.1)	0.365 (-5.9)	27.49 (-0.02)
0.49971	0.737	31.22	0.49984 (+0.0)	0.667 (-9.6)	31.17 (-0.05)
0.99917	1.402	36.06	0.99962 (+0.0)	1.170 (-16.5)	35.99 (-0.08)
1.99823	2.618	42.26	1.99861 (+0.0)	1.849 (-29.4)	42.17 (-0.08)

To conclude this section, it is worthwhile comparing the lossless performance associated with the various different modifications and options which have been advanced. Table 2 provides this comparison in terms

of lossless bit-rate and total number of arithmetically coded symbols per sample, for five different algorithms.

Table 2 *Comparison of lossless coding performance for five different algorithms (mode variations) using 64x64 code-blocks and 16x16 sub-blocks where applicable with the 5/3 default reversible Wavelet kernel. The first pair of columns refer to VM4 (EBCOT); the second pair refer to the coder obtained by applying the modifications described in Sections II-1 and II-2; the third pair of columns is obtained by adding the parallel options ("-Crestart" and sub-block causal context formation); the fourth pair of columns are obtained by adding the "lazy" coding option; and the last pair of columns are obtained using the "option" coder from VM4 (EBCOT) with "-Ccausal".*

VM4 bpp	VM4 Syms	Mod bpp	Mod Syms	Par bpp	Par Syms	Lazy bpp	Lazy Syms	Option bpp	Option Syms
Lenna (512x512)									
4.3024	4.87	4.3087	4.80	4.3344	4.80	4.3078	2.43	4.3553	5.19
Aerial2 (2048x2048)									
5.4502	5.68	5.4495	5.81	5.4700	5.81	5.3874	3.20	5.4793	6.02
Bike (2048x2560)									
4.5339	5.36	4.5373	5.27	4.5677	5.26	4.5667	2.37	4.5993	5.65
Cafe (2048x2560)									
5.3557	6.17	5.3598	6.07	5.3967	6.06	5.3527	2.59	5.4314	6.37
Woman (2048x2560)									
4.5158	5.11	4.5171	5.04	4.5429	5.04	4.5171	2.67	4.5758	5.36
Average (Bike, Café, Woman)									
4.8018	5.55	4.8047	5.46	4.8358	5.45	4.8122	2.54	4.8688	5.80

Some of the interesting points to observe are as follows:

- The "lazy" coding mode requires far fewer (usually less than half as many) symbols to be coded than any of the other modes.
- The "lazy" coding mode generates a lower compressed bit-rate than that obtained with the same parallel options but the lazy mode turned off. Since the lazy mode affects only the significance propagation and magnitude refinement coding passes all of whose coding contexts are initialized to the MQ coder's standard initial state at the beginning of the "learning curve", this result indicates that the relevant distributions must be so close to uniform that emitting raw binary digits is more efficient than letting the arithmetic coder learn this uniform distribution.
- The modifications to the original EBCOT algorithm described in Sections II-1 and II-2. have negligible effect on lossless performance.

Attachment.

REDUCED COMPLEXITY ENTROPY CODING WITH SUB-BLOCKS

Partners: UNSW, CISRA, HP, U.Arizona/SAIC, CRF, Sharp Laboratories of America

I INTRODUCTION

In this core experiment we explore the complexity-performance space associated with the EBCOT block coding engine [WG1N1020R] in VM4. VM4 currently contains two embedded block entropy coding engines: the original entropy coder from the EBCOT algorithm which provides the framework for VM4; and an "option" coder [WG1N1201], which provides an alternative, closely related entropy coding engine for the same framework. In some respects, the "option" coder involves some relatively minor modifications of the EBCOT entropy coder: elimination of one of the four EBCOT coding passes; and no transposition of code-blocks from the HL band. On the other hand, the "option" coder introduces some additional options which may have significant benefit to efficient hardware implementations: i.e. optional modes to enable parallel encoding and/or decoding and the possibility of implementations with reduced external memory consumption. On the other hand, the "option" coder also eliminated the use of sub-blocks within each code-block and the associated sub-block significance coder from the EBCOT algorithm. In fact the "option" coder was implemented by forcing the sub-block size in the EBCOT algorithm equal to the code-block size. During the Seoul meeting it became apparent to the authors that the sub-block framework need not be eliminated in order to realize the benefits available from the various modes offered by the "option" coder. In fact, with the aid of sub-blocks, we conjectured that it should be possible to maintain a lower implementation complexity and/or CPU execution time, while reducing some of the performance penalty associated with the "option" coder. This report and the associated source code represent the result of our investigations into these issues. Specifically, the goal here is to quantify the impact of a variety of incremental modifications and/or mode switches on the original EBCOT algorithm and to recommend a cohesive block-based entropy coding algorithm which possesses a superior range of complexity performance trade-offs to those which exist in VM4 today. In so doing, we put forth a recommendation which merges the best features of the two entropy coding algorithms in VM4.

It should be stressed that there are few innovations in this work. In fact, the similarities between the various entropy coding algorithms that we are considering far outweigh their differences. This is also evident from the similarity between their implementations which means that we are proposing only relatively minor changes to the VM4 software. The purpose of this work is thus to assemble a body of empirical evidence to enable us to settle on the fine details of the entropy coding algorithm as we approach the point of no return in the JPEG2000 standardization process. Specifically, we are concerned primarily with minimizing the following three sources of complexity, while sacrificing as little as possible in compression performance.

a) The total number of symbols which must be arithmetically encoded/decoded per original image pixel. While this figure does embody all aspects of complexity, it does represent a substantial portion of the implementation cost, particularly for hardware solutions. In this arena we explore mechanisms which reduce the average number of symbols which must be coded as well as mechanisms for reducing the maximum number of symbols which must be coded for any given subband sample.

b) Opportunities for parallelism. The independent block coding principle in EBCOT ensures that separate code-blocks can always be encoded/decoded in parallel, which we might understand as "macroscopic" parallelism. This type of parallelism is ideally suited to software implementations on multi-processor architectures and provides a simple, if expensive mechanism for increasing throughput in hardware implementations. Here we explore opportunities for "microscopic" parallelism along the lines of the "option" coder [WG1N1201], where the arithmetic coding contexts and codeword generation state variables are reset at the boundaries of each coding pass. This microscopic parallelism provides a mechanism for improving throughput without replicating the resources required to store multiple code-blocks in local memory. Microscopic parallelism of this form is probably most relevant for efficient hardware implementations.

c) Elimination of redundant coding passes and/or coding contexts whose cost in terms of implementation complexity does not justify the small gains in compression performance which result.

As one might expect, the above goals lead to a family of trade-offs between performance and complexity and it is important that we consider only the most compatible points in this complexity-performance trade-off so as to ensure that hardware and software implementations can support the full range of trade-offs (probably with different throughputs and/or power consumption) with comparative ease.

It turns out that the sub-block coding paradigm in EBCOT is of some significant assistance in containing the number of symbols which must be coded and also practical CPU execution times, which is one of the reasons for its original introduction in the entropy coding engine in VM4. On the other hand, there have been some questions regarding whether or not the quad-tree approach to coding the sub-block significance information is appropriate for JPEG2000. For this reason, we also investigate alternative approaches to coding the sub-block significance information.

In discussing the various modifications made to the EBCOT entropy coder in the following section, we provide summary comparative results in terms of the following five images:

- Lenna (512x512) -- this is included because it is the most common image used for comparing compression performance in journal papers and because it is a small image with which quick comparisons can be generated by others. It is not included in the averages.
- Aerial2 (2048x2048) -- one of the four most commonly used JPEG2000 test images. This image rarely shows substantial performance differences between different entropy coding algorithms, mainly because it is so noisy that schemes to exploit structure in the image are relatively meritless. We do not include this image in the averages because it invariably biases all entropy coding comparisons toward 0 difference.
- Bike (2048x2560) -- one of the four most commonly used JPEG2000 test images. This is included in the average.
- Café (2048x2560) -- one of the four most commonly used JPEG2000 test images. This is included in the average.
- Woman (2048x2560) -- one of the four most commonly used JPEG2000 test images. This is included in the average.

The 6-layer scalable mode is used with Daubechies 9/7 non-reversible filter kernels. This mode should by now be familiar to most readers. It is obtained with the following compression options, "-prof sp -Clayers 0.0625 0.125 0.25 0.5 1.0 -rate 2.0" along with whatever other arguments are required to enable specific options for the test. These incremental results which appear in this report do not necessarily fulfill the requirements for reporting core experiment results. A separate spreadsheet will be attached to fulfill that requirement, reporting more comprehensive test results only for the key modes whose recommendation comes out of this report. A discussion of the separate results to be reported on attached spreadsheets appears in Section IV.

We conclude this section by pointing out that the framework in which our implementation has been made and all associated results reported is that of a pre-release of the VM4.1 software, which is more up-to-date than any of the beta releases, but not current with the actual VM4.1 release. There are two important differences between the code we have worked with and VM4.1:

1) We have not used the new syntax by Ricoh, which became mandatory with the final release of VM4.1. This avoids the main difficulty with the new syntax in that it does not easily allow new quantities to be recorded in the global header of the bit-stream for ease of experimentation.
2) The "option" coder was actually changed immediately before the release of VM4.1 to use a different run-length coding method to that in the original EBCOT coder. In particular, this so-called "speed up" mode exploits properties of the arithmetic coding procedure (which has been modified accordingly) to

encode and/or decode multiple symbols at a time under some circumstances. Unfortunately, this modification became available after most of the results for this report had been generated. In fact, we summarize complexity in this report almost entirely in terms of the number of symbols which are coded, since the "option" coder used exactly the same pimitives as the EBCOT coder. With the "speed up" mode, the primitives are not identical and the symbol counting procedure has been broken (symbols are not counted at all within the "run" mode). It is not clear how symbols should be counted in the run mode and it is also not clear whether this mode should be added to the coder proposed in this document. For this reason, we choose to rely upon the earlier incarnation of the "option" coder for all the symbol count results presented here; however, to provide as much information as possible concerning the performance of the current version of the "option" coder, we provide comparative CPU execution times with the latest version in Section III. This appears to be the best way to deal with the fact that there is no reliable way to count symbols for the "fast" run mode and this mode has not yet been investigated within the setting of sub-block based coding.

II A SEQUENCE OF MODIFICATIONS TO THE EBCOT CODER

We begin with the EBCOT entropy coding engine from VM4 and express the entropy coding variations investigated in terms of a sequence of incremental modifications of this coder, along with the corresponding experimental results. In this way, we avoid the complexities of describing the entropy coder in detail. Only the differences from the coder described in [WG1N1020R] are actually identified here.

II-1 Preliminary Simplifications and Optimizations

We begin by eliminating some of the less useful features of the EBCOT entropy coder which contribute somewhat toward complexity.

II-1.1 No "Far" neighbours

Firstly, we eliminate the so-called "Far Neighbourhood" context state which is used by the zero coding primative when all 8 immediate neighbouring samples within the code-block are insignificant, but one or more of the six neighbours which reside 2 samples to the left and/or right and within 1 sample vertically is already significant. This eliminates one of the 10 zero coding model contexts and reduces complexity somewhat. It has no apparent effect on software running time although it does introduce some simplifications for hardware implementations since the coding context now depends on 8 rather than 14 neighbours and there is one less context to be stored (contexts might need to be kept in very fast register storage, which is much more expensive than on-chip SRAM). The effect on PSNR performance is on the order of about 0.01 dB and not worth reporting here.

II-1.2 Reduction from 4 to 3 Coding Passes per Bit-Plane

The original EBCOT embedded block coder specifies 4 coding passes per bit-plane, designated as follows:

- **"Forward significance propagation pass"**, $P_i^{p,1}$. We preserve this as is, except that we collect all subband samples for which any of the eight immediate neighbours have already been found to be significant into this pass to compensate for the fact that we are dropping the reverse pass, below.
- **"Backward significance propagation pass"**, $P_i^{p,2}$. We eliminate this coding pass.
- **"Magnitude refinement pass"**, $P_i^{p,3}$. We preserve this as is.
- **"Normalization pass"**, $P_i^{p,4}$. We preserve this as is.

In all other respects, the description in [WG1N1020R] remains unchanged. The reason for dropping the reverse pass is that it contributes very little to overall compression performance with most images and adds to the implementation complexity and software running time (although not to the number of symbols which must be coded). It is also partly incompatible with the options discussed in Section II-3 to increase opportunities for parallelism, although this could have been circumvented by using a reverse scan only

within sub-blocks, if it had turned out to be sufficiently important. Table 3 shows the comparison between the performance of VM4 and the modified coder obtained by removing the backward significance propagation pass and the "Far" neighbourhood context.

Table 3 *Comparison of VM4 with coder obtained by introducing modifications described in Sections II-1.1 and II-1.2*

Lenna (512x512)

VM4 bpp	VM4 symbols	VM4 PSNR	Mod bpp (+/- %)	Mod Symbols (+/- %)	Mod PSNR (+/- dB)
0.06174	0.093	28.19	0.06229 (+0.9)	0.091 (-2.2)	28.21 (+0.02)
0.12482	0.189	31.10	0.12433 (-0.4)	0.180 (-4.8)	31.06 (-0.04)
0.24982	0.382	34.20	0.24899 (-0.3)	0.367 (-4.0)	34.17 (-0.03)
0.49905	0.749	37.33	0.49933 (+0.1)	0.720 (-3.8)	37.32 (-0.01)
0.99942	1.430	40.43	0.99994 (+0.1)	1.378 (-3.6)	40.44 (+0.00)
1.99094	2.590	44.92	1.99579 (+0.2)	2.513 (-3.0)	44.92 (-0.00)

Aerial 2 (2048x2048+A45

VM4 bpp	VM4 symbols	VM4 PSNR	Mod bpp (+/- %)	Mod Symbols (+/- %)	Mod PSNR (+/- dB)
0.06248	0.094	24.66	0.06230 (-0.3)	0.090 (-4.1)	24.63 (-0.03)
0.12484	0.192	26.53	0.12441 (-0.3)	0.185 (-3.6)	26.51 (-0.02)
0.24995	0.364	28.58	0.24999 (+0.0)	0.350 (-3.8)	28.58 (-0.00)
0.49934	0.694	30.63	0.49982 (+0.1)	0.674 (-3.0)	30.63 (-0.00)
0.99969	1.349	33.27	0.99930 (-0.0)	1.303 (-3.4)	33.24 (-0.02)
1.99812	2.514	38.11	1.99850 (+0.0)	2.445 (-2.7)	38.10 (-0.01)

Bike (2048x2560)

VM4 bpp	VM4 symbols	VM4 PSNR	Mod bpp (+/- %)	Mod Symbols (+/- %)	Mod PSNR (+/- dB)
0.06215	0.120	23.81	0.06249 (+0.5)	0.115 (-3.4)	23.80 (-0.02)
0.12483	0.229	26.41	0.12496 (+0.1)	0.220 (-3.9)	26.36 (-0.05)
0.24985	0.440	29.66	0.24994 (+0.0)	0.423 (-3.9)	29.62 (-0.04)
0.49941	0.836	33.54	0.49986 (+0.1)	0.802 (-4.1)	33.51 (-0.03)
0.99997	1.552	38.12	0.99972 (-0.0)	1.488 (-4.1)	38.08 (-0.03)
1.99589	2.825	44.03	1.99894 (+0.2)	2.723 (-3.6)	44.03 (-0.01)

Café (2048x2560)

VM4 bpp	VM4 symbols	VM4 PSNR	Mod bpp (+/- %)	Mod Symbols (+/- %)	Mod PSNR (+/- dB)
0.06246	0.101	19.07	0.06240 (-0.1)	0.098 (-3.2)	19.05 (-0.02)
0.12493	0.212	20.81	0.12462 (-0.2)	0.204 (-3.6)	20.75 (-0.06)
0.24985	0.405	23.16	0.24971 (-0.1)	0.389 (-4.1)	23.13 (-0.03)
0.49957	0.765	26.83	0.49982 (+0.1)	0.738 (-3.5)	26.80 (-0.03)
0.99999	1.485	32.06	0.99995 (-0.0)	1.427 (-3.9)	32.03 (-0.03)
1.99883	2.759	39.12	1.99894 (+0.0)	2.661 (-3.6)	39.07 (-0.05)

Woman (2048x2560)

VM4 bpp	VM4 symbols	VM4 PSNR	Mod bpp (+/- %)	Mod Symbols (+/- %)	Mod PSNR (+/- dB)
0.06222	0.094	25.63	0.06248 (+0.4)	0.091 (-2.9)	25.63 (+0.00)
0.12490	0.193	27.39	0.12471 (-0.2)	0.185 (-4.2)	27.36 (-0.03)
0.24983	0.373	30.02	0.24984 (+0.0)	0.359 (-4.0)	30.01 (-0.01)
0.49965	0.711	33.66	0.49896 (-0.1)	0.685 (-3.6)	33.63 (-0.03)
0.99998	1.367	38.45	0.99964 (-0.0)	1.324 (-3.1)	38.44 (-0.01)
1.99885	2.598	44.05	1.99980 (+0.0)	2.521 (-3.0)	44.04 (-0.01)

Average (Bike, Café, Woman)

VM4 bpp	VM4 symbols	VM4 PSNR	Mod bpp (+/- %)	Mod Symbols (+/- %)	Mod PSNR (+/- dB)
0.06228	0.105	22.84	0.06246 (+0.3)	0.101 (-3.2)	22.82 (-0.01)
0.12489	0.211	24.87	0.12476 (-0.1)	0.203 (-3.9)	24.83 (-0.04)
0.24985	0.406	27.61	0.24983 (-0.0)	0.390 (-4.0)	27.59 (-0.03)
0.49954	0.771	31.34	0.49955 (+0.0)	0.742 (-3.7)	31.31 (-0.03)
0.99998	1.468	36.21	0.99977 (-0.0)	1.413 (-3.7)	36.18 (-0.02)
1.99786	2.727	42.40	1.99923 (+0.1)	2.635 (-3.4)	42.38 (-0.02)

II-1.3 No Block Transposition

In the original EBCOT entropy coder, each code-block from the HL subband (horizontally high-pass, vertically low-pass) subband is transposed so that both LH and HL code-blocks (the most significant) contain predominantly vertical edge features. This somewhat simplified the implementation and testing in software and also reduced the complexity when the "Far Neighbourhood" context was in use. On the other hand, if we remove the "Far" neighbourhood context, as described above, this transposition step could do more harm than good to the overall complexity. More importantly, the transposition operation prevents us from deriving implementations with reduced external memory consumption as discussed in Section II-3. The penalty for eliminating transposition is that the run-length coding mode in EBCOT might not respond so well to the HL subband's code-blocks so that we might expect poorer performance. Table 4, however, shows that this impact is negligible.

Table 4 *Comparison of VM4 with coder obtained by introducing modifications described in Sections II-1.1, II-1.2 and II-1.3.*

Lenna (512x512)

VM4 bpp	VM4 symbols	VM4 PSNR	Mod bpp (+/-%)	Mod Symbols (+/-%)	Mod PSNR (+/-dB)
0.06174	0.093	28.19	0.06152 (-0.3)	0.091 (-2.3)	28.18 (-0.01)
0.12482	0.189	31.10	0.12488 (+0.0)	0.183 (-3.1)	31.06 (-0.05)
0.24982	0.382	34.20	0.24915 (-0.3)	0.372 (-2.8)	34.16 (-0.04)
0.49905	0.749	37.33	0.49942 (+0.1)	0.729 (-2.7)	37.30 (-0.03)
0.99942	1.430	40.43	0.99954 (+0.0)	1.388 (-2.9)	40.42 (-0.02)
1.99094	2.590	44.92	1.99225 (+0.1)	2.521 (-2.7)	44.88 (-0.04)

Aerial 2 (2048x2048+A45)

VM4 bpp	VM4 symbols	VM4 PSNR	Mod bpp (+/-%)	Mod Symbols (+/-%)	Mod PSNR (+/-dB)
0.06248	0.094	24.66	0.06246 (-0.0)	0.090 (-3.6)	24.63 (-0.02)
0.12484	0.192	26.53	0.12425 (-0.5)	0.185 (-3.3)	26.50 (-0.03)
0.24995	0.364	28.58	0.25000 (+0.0)	0.351 (-3.4)	28.58 (-0.00)
0.49934	0.694	30.63	0.49978 (+0.1)	0.675 (-2.7)	30.63 (-0.00)
0.99969	1.349	33.27	0.99996 (+0.0)	1.306 (-3.2)	33.25 (-0.02)
1.99812	2.514	38.11	1.99762 (-0.0)	2.446 (-2.7)	38.10 (-0.01)

Bike (2048x2560)

VM4 bpp	VM4 symbols	VM4 PSNR	Mod bpp (+/-%)	Mod Symbols (+/-%)	Mod PSNR (+/-dB)
0.06215	0.120	23.81	0.06245 (+0.5)	0.118 (-1.5)	23.79 (-0.03)
0.12483	0.229	26.41	0.12486 (+0.0)	0.224 (-2.4)	26.34 (-0.07)
0.24985	0.440	29.66	0.24978 (-0.0)	0.429 (-2.5)	29.61 (-0.05)
0.49941	0.836	33.54	0.49998 (+0.1)	0.812 (-2.9)	33.50 (-0.03)
0.99997	1.552	38.12	0.99989 (-0.0)	1.504 (-3.1)	38.08 (-0.03)
1.99589	2.825	44.03	1.99885 (+0.1)	2.739 (-3.0)	44.02 (-0.01)

Café (2048x2560)

VM4 bpp	VM4 symbols	VM4 PSNR	Mod bpp (+/-%)	Mod Symbols (+/-%)	Mod PSNR (+/-dB)
0.06246	0.101	19.07	0.06230 (-0.3)	0.099 (-2.0)	19.04 (-0.02)
0.12493	0.212	20.81	0.12458 (-0.3)	0.207 (-2.2)	20.75 (-0.06)
0.24985	0.405	23.16	0.24995 (+0.0)	0.395 (-2.5)	23.13 (-0.03)
0.49957	0.765	26.83	0.50000 (+0.1)	0.748 (-2.2)	26.79 (-0.03)
0.99999	1.485	32.06	0.99928 (-0.1)	1.441 (-2.9)	32.02 (-0.05)
1.99883	2.759	39.12	1.99925 (+0.0)	2.680 (-2.9)	39.07 (-0.05)

Woman (2048x2560)

VM4 bpp	VM4 symbols	VM4 PSNR	Mod bpp (+/-%)	Mod Symbols (+/-%)	Mod PSNR (+/-dB)
0.06222	0.094	25.63	0.06243 (+0.3)	0.092 (-2.7)	25.62 (-0.01)
0.12490	0.193	27.39	0.12462 (-0.2)	0.186 (-3.7)	27.35 (-0.03)
0.24983	0.373	30.02	0.24986 (+0.0)	0.360 (-3.7)	30.00 (-0.02)
0.49965	0.711	33.66	0.49977 (+0.0)	0.688 (-3.2)	33.63 (-0.02)
0.99998	1.367	38.45	1.00000 (+0.0)	1.329 (-2.8)	38.43 (-0.01)
1.99885	2.598	44.05	1.99844 (-0.0)	2.526 (-2.8)	44.03 (-0.02)

Average (Bike, Café, Woman)

VM4 bpp	VM4 symbols	VM4 PSNR	Mod bpp (+/-%)	Mod Symbols (+/-%)	Mod PSNR (+/-dB)
0.06228	0.105	22.84	0.06239 (+0.2)	0.103 (-2.0)	22.82 (-0.02)
0.12489	0.211	24.87	0.12469 (-0.2)	0.206 (-2.7)	24.81 (-0.06)
0.24985	0.406	27.61	0.24987 (+0.0)	0.395 (-2.9)	27.58 (-0.04)
0.49954	0.771	31.34	0.49992 (+0.1)	0.750 (-2.7)	31.31 (-0.03)
0.99998	1.468	36.21	0.99972 (-0.0)	1.425 (-2.9)	36.18 (-0.03)
1.99786	2.727	42.40	1.99885 (+0.0)	2.649 (-2.9)	42.37 (-0.03)

II-1.4 Skewed Initialization for the Arithmetic Coder

The original EBCOT coder uses the default initial state for all context models in the conditional arithmetic coding process. Some benefit can be gained by initializing the run-length and all-zero contexts with appropriately skewed distributions, as was done for the option coder in [WG1N1201]. Also, for symbols with an assumed uniform distribution the MQ coder introduced from Seoul uses a separate context, which was inappropiately initialized in VM4. Nevertheless, preferential initialization of the arithmetic coder context states has the most substantial effect when the coder is restarted at the beginning of each coding pass as described in Section II-3. We take the tuned initial states for this case and apply them to the coder with the modifications described in the previous three sections so as to move toward the starting point in a family of very closely related coders which is developed in the following sections. The initializations themselves are shown below; the results shown in Table 5 indicate that the initializations which are tuned for independent coding passes have no substantial impact when the coding passes are not independent.

Run-length coding context (for runs of four samples, all currently insignificant, with all neighbours of each sample currently insignificant)	MQ coder state 6 (Probability of "1" = 0.042; on the "learning curve")
All zero neighbourhood context (for samples which are currently insignificant, with all neighbours insignificant, which do not qualify as members of a run)	MQ coder state 8 (Probability of "1" = 0.020; on the "learning curve")
Assumed uniform context (for symbols which were coded without any adaptive context prior to the introduction of the MQ coder)	MQ coder state 28 (Probability of "1" = 0.34; not on the "learning curve") This context is re-initialized at the start of each coding pass.
All other context models	MQ coder state 0 (Probability of "1" = 0.34; at start of "learning curve")

Table 5 *Comparison of VM4 with coder obtained by introducing modifications described in Sections II-1.1, II-1.2, II-1.3 and II-1.4.*

Lenna (512x512)

VM4 bpp	VM4 symbols	VM4 PSNR	Mod bpp (+/-%)	Mod Symbols (+/-%)	Mod PSNR (+/-dB)
0.06174	0.093	28.19	0.06171 (-0.1)	0.091 (-2.3)	28.18 (-0.01)
0.12482	0.189	31.10	0.12491 (+0.1)	0.183 (-3.1)	31.06 (-0.05)
0.24982	0.382	34.20	0.24973 (-0.0)	0.373 (-2.4)	34.17 (-0.03)
0.49905	0.749	37.33	0.49976 (+0.1)	0.730 (-2.6)	37.31 (-0.02)
0.99942	1.430	40.43	0.99973 (+0.0)	1.388 (-2.9)	40.41 (-0.03)
1.99094	2.590	44.92	1.99149 (+0.0)	2.521 (-2.7)	44.88 (-0.04)

Aerial 2 (2048x2048+A45

VM4 bpp	VM4 symbols	VM4 PSNR	Mod bpp (+/-%)	Mod Symbols (+/-%)	Mod PSNR (+/-dB)
0.06248	0.094	24.66	0.06248 (+0.0)	0.091 (-3.5)	24.63 (-0.02)
0.12484	0.192	26.53	0.12490 (+0.1)	0.186 (-3.0)	26.51 (-0.02)
0.24995	0.364	28.58	0.24993 (-0.0)	0.352 (-3.4)	28.58 (-0.00)
0.49934	0.694	30.63	0.49986 (+0.1)	0.676 (-2.7)	30.63 (-0.00)
0.99969	1.349	33.27	0.99996 (+0.0)	1.307 (-3.1)	33.25 (-0.01)
1.99812	2.514	38.11	1.99737 (-0.0)	2.446 (-2.7)	38.10 (-0.01)

Bike (2048x2560)

VM4 bpp	VM4 symbols	VM4 PSNR	Mod bpp (+/-%)	Mod Symbols (+/-%)	Mod PSNR (+/-dB)
0.06215	0.120	23.81	0.06242 (+0.4)	0.118 (-1.6)	23.79 (-0.03)
0.12483	0.229	26.41	0.12465 (-0.1)	0.224 (-2.5)	26.34 (-0.08)
0.24985	0.440	29.66	0.24975 (-0.0)	0.428 (-2.6)	29.61 (-0.05)
0.49941	0.836	33.54	0.49991 (+0.1)	0.812 (-2.9)	33.50 (-0.03)
0.99997	1.552	38.12	0.99992 (-0.0)	1.504 (-3.1)	38.08 (-0.03)
1.99589	2.825	44.03	1.99944 (+0.2)	2.741 (-3.0)	44.03 (-0.01)

Café (2048x2560)

VM4 bpp	VM4 symbols	VM4 PSNR	Mod bpp (+/-%)	Mod Symbols (+/-%)	Mod PSNR (+/-dB)
0.06246	0.101	19.07	0.06230 (-0.3)	0.099 (-1.9)	19.05 (-0.02)
0.12493	0.212	20.81	0.12462 (-0.3)	0.207 (-2.2)	20.75 (-0.06)
0.24985	0.405	23.16	0.24948 (-0.1)	0.394 (-2.7)	23.12 (-0.04)
0.49957	0.765	26.83	0.49992 (+0.1)	0.748 (-2.2)	26.79 (-0.04)
0.99999	1.485	32.06	0.99906 (-0.1)	1.441 (-2.9)	32.02 (-0.05)
1.99883	2.759	39.12	1.99651 (-0.1)	2.678 (-2.9)	39.06 (-0.06)

Woman (2048x2560)

VM4 bpp	VM4 symbols	VM4 PSNR	Mod bpp (+/-%)	Mod Symbols (+/-%)	Mod PSNR (+/-dB)
0.06222	0.094	25.63	0.06233 (+0.2)	0.091 (-2.9)	25.62 (-0.01)
0.12490	0.193	27.39	0.12483 (-0.1)	0.187 (-3.4)	27.36 (-0.03)
0.24983	0.373	30.02	0.24971 (-0.0)	0.360 (-3.7)	30.00 (-0.03)
0.49965	0.711	33.66	0.49993 (+0.1)	0.688 (-3.2)	33.63 (-0.03)
0.99998	1.367	38.45	0.99989 (-0.0)	1.329 (-2.8)	38.43 (-0.02)
1.99885	2.598	44.05	1.99481 (-0.2)	2.523 (-2.9)	44.02 (-0.03)

Average (Bike, Café, Woman)

VM4 bpp	VM4 symbols	VM4 PSNR	Mod bpp (+/-%)	Mod Symbols (+/-%)	Mod PSNR (+/-dB)
0.06228	0.105	22.84	0.06235 (+0.1)	0.103 (-2.1)	22.82 (-0.02)
0.12489	0.211	24.87	0.12470 (-0.2)	0.206 (-2.7)	24.82 (-0.06)
0.24985	0.406	27.61	0.24965 (-0.1)	0.394 (-3.0)	27.57 (-0.04)
0.49954	0.771	31.34	0.49992 (+0.1)	0.750 (-2.7)	31.31 (-0.03)
0.99998	1.468	36.21	0.99962 (-0.0)	1.425 (-2.9)	36.18 (-0.03)
1.99786	2.727	42.40	1.99692 (-0.0)	2.647 (-2.9)	42.37 (-0.04)

II-2 Alternatives to Quad-Tree Coding of Sub-Block Significance

In the original EBCOT algorithm the significance of each sub-block within a code-block is coded using a simple embedded quad-tree code. Here we develop an alternative method based on the techniques used for bit-plane coding. Specifically, at the beginning of each bit-plane, we use the arithmetic coding engine rather than a quad-tree code to signal the significance of each sub-block relative to that bit-plane. We simply scan through all the sub-blocks in the code-block in the usual lexicographical order, skipping over those sub-blocks which are already known to be significant and emitting a single binary symbol to identify the significance of each of the other blocks ("1" if it becomes significant, "0" otherwise). The symbol is coded in one of two different contexts as follows:

- If any of the four immediate neighbours (above, below, to the left or to the right) of the sub-block which lie within the same code-block have already been found to be significant, we use the "assumed uniform" coding context, which is reinitialized to the usual value (MQ coder state 28) at the beginning of the scan; remember that this context is reinitialized at the start of every coding pass, so reusing for the quad-tree code does not interfere with any of the other coding operations.
- Otherwise, when all four immediate neighbours are still insignificant, we use a special coding context which is initialized to MQ coder state 4 (a slightly skewed state near the start of the "learning curve") at the commencement of the scan.

The bit-plane coding operations are interleaved into the bit-stream in exactly the same manner and sub-block significance has exactly the same interpretation as in the original EBCOT algorithm described in [WG1N1020R]. We find that this algorithm gives essentially the same coding performance as the quad-tree coder, with a negligible improvement of between 0.01dB and 0.02dB. Table 6 provides comparative performance figures for the quad-tree coder and the so-called scan-subs coder in the default case of 16x16 sub-blocks and 64x64 code-blocks. Of more interest is the comparison shown in Table 7 for 8x8 sub-blocks and 64x64 code-blocks, since in this case the sub-block significance code comprises a greater proportion of the overall bit-rate.

Table 6 *Comparison of the embedded quad-tree with the scan-subs method for coding sub-block significance. In both cases, the modifications described in Sections II-1.1, II-1.2, II-1.3 and II-1.4 are included. Here the default sub-block and code-block sizes of 16x16 and 64x64 respectively are used.*

Lenna (512x512)

Quad bpp	Quad symbols	Quad PSNR	Scan bpp (+/-%)	Scan Symbols (+/-%)	Scan PSNR (+/-dB)
0.06171	0.091	28.18	0.06146 (-0.4)	0.091 (+0.1)	28.18
0.12491	0.183	31.06	0.12497 (+0.0)	0.186 (+1.9)	31.08 (+0.03)
0.24973	0.373	34.17	0.24973	0.374 (+0.1)	34.17 (+0.00)
0.49976	0.730	37.31	0.49954 (-0.0)	0.732 (+0.3)	37.32 (+0.01)
0.99973	1.388	40.41	0.99829 (-0.1)	1.389 (+0.1)	40.41
1.99149	2.521	44.88	1.99039 (-0.1)	2.522 (+0.0)	44.88 (-0.00)

Aerial 2 (2048x2048+A45

Quad bpp	Quad symbols	Quad PSNR	Scan bpp (+/-%)	Scan Symbols (+/-%)	Scan PSNR (+/-dB)
0.06248	0.091	24.63	0.06220 (-0.5)	0.091 (+0.7)	24.63 (-0.01)
0.12490	0.186	26.51	0.12487 (-0.0)	0.187 (+0.4)	26.52 (+0.01)
0.24993	0.352	28.58	0.24999 (+0.0)	0.353 (+0.3)	28.58 (+0.01)
0.49986	0.676	30.63	0.49994 (+0.0)	0.677 (+0.2)	30.63 (+0.00)
0.99996	1.307	33.25	0.99913 (-0.1)	1.308 (+0.1)	33.25 (+0.00)
1.99737	2.446	38.10	1.99895 (+0.1)	2.449 (+0.1)	38.11 (+0.01)

Bike (2048x2560)

Quad bpp	Quad symbols	Quad PSNR	Scan bpp (+/-%)	Scan Symbols (+/-%)	Scan PSNR (+/-dB)
0.06242	0.118	23.79	0.06241 (-0.0)	0.119 (+0.7)	23.81 (+0.02)
0.12465	0.224	26.34	0.12451 (-0.1)	0.225 (+0.5)	26.35 (+0.01)
0.24975	0.428	29.61	0.24995 (+0.1)	0.431 (+0.6)	29.63 (+0.02)
0.49991	0.812	33.50	0.49994 (+0.0)	0.816 (+0.4)	33.52 (+0.02)
0.99992	1.504	38.08	0.99980 (-0.0)	1.508 (+0.3)	38.09 (+0.01)
1.99944	2.741	44.03	1.99945 (+0.0)	2.743 (+0.1)	44.03 (+0.01)

Café (2048x2560)

Quad bpp	Quad symbols	Quad PSNR	Scan bpp (+/-%)	Scan Symbols (+/-%)	Scan PSNR (+/-dB)
0.06230	0.099	19.05	0.06240 (+0.2)	0.100 (+1.0)	19.06 (+0.01)
0.12462	0.207	20.75	0.12460 (-0.0)	0.208 (+0.6)	20.76 (+0.01)
0.24948	0.394	23.12	0.24990 (+0.2)	0.396 (+0.5)	23.14 (+0.02)
0.49992	0.748	26.79	0.49960 (-0.1)	0.750 (+0.2)	26.80 (+0.01)
0.99906	1.441	32.02	0.99953 (+0.0)	1.444 (+0.2)	32.03 (+0.01)
1.99651	2.678	39.06	1.99715 (+0.0)	2.680 (+0.1)	39.07 (+0.01)

Woman (2048x2560)

Quad bpp	Quad symbols	Quad PSNR	Scan bpp (+/-%)	Scan Symbols (+/-%)	Scan PSNR (+/-dB)
0.06233	0.091	25.62	0.06222 (-0.2)	0.092 (+0.4)	25.62 (+0.00)
0.12483	0.187	27.36	0.12496 (+0.1)	0.188 (+0.7)	27.37 (+0.01)
0.24971	0.360	30.00	0.24989 (+0.1)	0.362 (+0.5)	30.01 (+0.02)
0.49993	0.688	33.63	0.49974 (-0.0)	0.691 (+0.3)	33.64 (+0.01)
0.99989	1.329	38.43	0.99993 (+0.0)	1.332 (+0.2)	38.44 (+0.01)
1.99481	2.523	44.02	1.99878 (+0.2)	2.529 (+0.2)	44.04 (+0.02)

Average (Bike, Café, Woman)

Quad bpp	Quad symbols	Quad PSNR	Scan bpp (+/-%)	Scan Symbols (+/-%)	Scan PSNR (+/-dB)
0.06235	0.103	22.82	0.06235 (-0.0)	0.103 (+0.7)	22.83 (+0.01)
0.12470	0.206	24.82	0.12469 (-0.0)	0.207 (+0.6)	24.83 (+0.01)
0.24965	0.394	27.57	0.24991 (+0.1)	0.396 (+0.6)	27.59 (+0.02)
0.49992	0.750	31.31	0.49976 (-0.0)	0.752 (+0.3)	31.32 (+0.01)
0.99962	1.425	36.18	0.99975 (+0.0)	1.428 (+0.2)	36.19 (+0.01)
1.99692	2.647	42.37	1.99846 (+0.1)	2.651 (+0.1)	42.38 (+0.01)

Table 7 *Comparison of the embedded quad-tree with the scan-subs method for coding sub-block significance. In both cases, the modifications described in Sections II-1.1, II-1.2, II-1.3 and II-1.4 are included. Here the sub-block and code-block sizes are 8x8 and 64x64, respectively.*

Lenna (512x512)

Quad bpp	Quad symbols	Quad PSNR	Scan bpp (+/-%)	Scan Symbols (+/-%)	Scan PSNR (+/-dB)
0.06232	0.080	28.18	0.06155 (-1.2)	0.081 (+1.0)	28.15 (-0.02)
0.12463	0.162	31.03	0.12473 (+0.1)	0.165 (+2.1)	31.05 (+0.02)
0.24921	0.329	34.14	0.24988 (+0.3)	0.337 (+2.5)	34.16 (+0.02)
0.49988	0.651	37.29	0.49902 (-0.2)	0.663 (+1.9)	37.30 (+0.01)
0.99912	1.267	40.40	0.99948 (+0.0)	1.283 (+1.3)	40.42 (+0.02)
1.99634	2.403	44.88	1.99454 (-0.1)	2.416 (+0.6)	44.88

Aerial 2 (2048x2048+A45

Quad bpp	Quad symbols	Quad PSNR	Scan bpp (+/-%)	Scan Symbols (+/-%)	Scan PSNR (+/-dB)
0.06187	0.081	24.59	0.06245 (+0.9)	0.083 (+2.2)	24.63 (+0.04)
0.12451	0.166	26.49	0.12414 (-0.3)	0.169 (+1.4)	26.49 (+0.00)
0.24996	0.325	28.56	0.24999 (+0.0)	0.329 (+1.3)	28.57 (+0.01)
0.49968	0.624	30.61	0.49963 (-0.0)	0.630 (+1.0)	30.62 (+0.01)
0.99977	1.235	33.24	0.99987 (+0.0)	1.243 (+0.6)	33.25 (+0.01)
1.99924	2.368	38.09	1.99770 (-0.1)	2.374 (+0.2)	38.09 (+0.00)

Bike (2048x2560)

Quad bpp	Quad symbols	Quad PSNR	Scan bpp (+/-%)	Scan Symbols (+/-%)	Scan PSNR (+/-dB)
0.06249	0.100	23.77	0.06250 (+0.0)	0.103 (+3.2)	23.79 (+0.02)
0.12497	0.192	26.31	0.12475 (-0.2)	0.197 (+2.7)	26.33 (+0.02)
0.24989	0.372	29.57	0.24950 (-0.2)	0.381 (+2.4)	29.58 (+0.01)
0.49943	0.718	33.46	0.49942 (-0.0)	0.731 (+1.8)	33.48 (+0.02)
0.99908	1.370	38.05	0.99941 (+0.0)	1.387 (+1.2)	38.06 (+0.02)
1.99950	2.571	44.00	1.99934 (-0.0)	2.587 (+0.6)	44.01 (+0.01)

Café (2048x2560)

Quad bpp	Quad symbols	Quad PSNR	Scan bpp (+/-%)	Scan Symbols (+/-%)	Scan PSNR (+/-dB)
0.062234	0.087333	19.0222	0.062369 (+0.2)	0.088741 (+1.6)	19.0387 (+0.02)
0.124915	0.181482	20.735	0.124591 (-0.3)	0.183589 (+1.2)	20.7435 (+0.01)
0.249942	0.353386	23.1011	0.249858 (-0.0)	0.357532 (+1.2)	23.117 (+0.02)
0.499956	0.681698	26.7642	0.499768 (-0.0)	0.688093 (+0.9)	26.7774 (+0.01)
0.999236	1.33668	31.9912	0.999153 (-0.0)	1.34688 (+0.8)	32.0058 (+0.01)
1.99996	2.53935	39.0549	1.99741 (-0.1)	2.54795 (+0.3)	39.0521 (-0.00)

Woman (2048x2560)

Quad bpp	Quad symbols	Quad PSNR	Scan bpp (+/-%)	Scan Symbols (+/-%)	Scan PSNR (+/-dB)
0.062492	0.081734	25.6227	0.062256 (-0.4)	0.083419 (+2.1)	25.6242 (+0.00)
0.124985	0.165791	27.3626	0.124609 (-0.3)	0.169562 (+2.3)	27.3658 (+0.00)
0.249225	0.326258	29.9812	0.249635 (+0.2)	0.332498 (+1.9)	30.0001 (+0.02)
0.49968	0.635982	33.6191	0.499887 (+0.0)	0.645346 (+1.5)	33.6351 (+0.02)
0.999327	1.24822	38.4185	0.999353 (+0.0)	1.2603 (+1.0)	38.4303 (+0.01)
1.99589	2.41545	44.0039	1.99937 (+0.2)	2.43304 (+0.7)	44.0289 (+0.02)

Average (Bike, Café, Woman)

Quad bpp	Quad symbols	Quad PSNR	Scan bpp (+/-%)	Scan Symbols (+/-%)	Scan PSNR (+/-dB)
0.06240	0.090	22.80	0.06237 (-0.0)	0.092 (+2.4)	22.82 (+0.01)
0.12496	0.180	24.80	0.12465 (-0.2)	0.183 (+2.0)	24.81 (+0.01)
0.24969	0.350	27.55	0.24967 (-0.0)	0.357 (+1.8)	27.57 (+0.02)
0.49969	0.678	31.28	0.49969 (+0.0)	0.688 (+1.4)	31.30 (+0.02)
0.99921	1.318	36.15	0.99931 (+0.0)	1.331 (+1.0)	36.17 (+0.01)
1.99845	2.509	42.35	1.99871 (+0.0)	2.523 (+0.6)	42.36 (+0.01)

II-3 Options to Enable Parallelism

As with any block coder, it is always possible to compress or decompress any number of blocks in parallel and indeed this is most likely the easiest way to exploit parallelism. We refer to this as macroscopic parallelism, because it does not require tight synchronization between the respective coding engines and can be realized in multi-threaded software implementations with comparative ease. A second opportunity for parallelism arises if the coding passes for any given block can also be performed in parallel. This is enabled if separate probability model contexts are maintained for each coding pass within each code-block. Of course, there is some additional adaptation overhead in this case, but this overhead is minimized by the fact that we only visit those sub-blocks which are known to be significant already. Moreover, we will need to terminate the arithmetic code word for each coding pass in such a way as to ensure that the decoder can recover the termination point implicitly during the decoding process and so know where each subsequent coding pass's bit-stream begins. This involves an average overhead of approximately 1.5 bits. Finally, if we want to enable parallel decoding, as well as encoding, the length of the bit-stream segment corresponding to each coding pass must be explicitly identified, rather than implicitly determined by the decoding process itself and we must also restrict the non-causal context formation model used in the original EBCOT algorithm. These conditions and their solution are virtually identical to those associated with the "option" coder in VM4 [WG1N1201], with the following exceptions:

- There is no need to depart from the sub-block paradigm, as done in the "option" coder, or to entirely abandon the non-causal context models. Instead, we obtain a solution with higher compression performance and a reduced number of symbols to be arithmetically encoded/decoded by modifying the coder described above so that the coding contexts are "sub-block causal". That is, no sample in one sub-block may be coded with respect to a context formed using samples from a later sub-block in the lexicographical scan pattern. This affects the coding contexts only for those samples which lie on the lower and right hand boundaries of each sub-block. In fact, the modification is so minor that our implementation does not need to invoke separate coding pass functions for this mode.
- The arithmetic coder is also terminated and restarted at the beginning of each sub-block significance coding pass using the algorithm outlined in Section II-2.

We currently envisage these modifications (sub-block causal context formation, restarting the arithmetic coder and reinitializing the context states at each coding pass, and markers to assist with parallel decoding) as options for the bit-stream since they each involve some sacrifice in compression performance and the reward can only be realized in certain hardware implementations. Nevertheless, the modifications represent minor departures from the first member of the proposed family of entropy coders, i.e. that obtained by applying all the modifications discussed in Sections II-1 and II-2. For convenience, we will identify the option to reset the context states and restart the arithmetic coder at each coding pass boundary as "-Crestart". Before providing some experimental results, we discuss the implications of these optional modifications.

II-3.1 Implications for Parallelism

Here we briefly consider some of the options for parallel decoding imparted by the sub-block causal coding context option. Parallel encoding relies only on the fact that the arithmetic coder is restarted on coding pass boundaries (i.e. "-Crestart"), although a parallel implementation might be substantially simplified by the sub-block causal coding contexts since then the encoder and decoder implementations can be virtually identical. Since the sub-blocks do not affect each other in a non-causal manner, it is clear that we can run parallel arithmetic coding engines in each sub-block, provided they are appropriately synchronized. Specifically, when the engine associated with the n'th sub-block in the scan completes a coding pass, it transfers its state and boundary conditions to the engine associated with the (n+1)'th sub-block in the scan which then performs the same coding pass, while the engine associated with the n'th sub-block receives new state and boundary information from the engine for the (n-1)'th sub-block in the scan. This is only one of a number of parallel implementation options, but is perhaps the easiest to understand. It should be noted that a parallel decoder must decode the sub-block significance information for all bit-planes (or all those to be executed in parallel) in a sequential manner before the parallel decoding of the sub-bitplane passes can commence. This presents no problem since the sub-block significance coding process is remarkably simple and operates on vastly less data than the regular coding passes.

II-3.2 Opportunities for Memory Reduction

In applications where external memory and memory bandwidth are at a premium and the application supplies or consumes the image in a line-by-line fashion, perhaps the most reasonable approach is to use a "K"-line transform implementation which produces K lines of each subband in any given resolution level in a swath, storing LL band samples back to memory until enough have been accumulated to run the transform over the same "K"-line transform over the LL band and so on in a recursive manner, exactly as explained in [WG1N1020R]. Larger values of K imply lower overall memory access bandwidth, since the cost of accessing the samples in the filter's vertical region of support is amortized over the larger number of lines being produced simultaneously. When working with the generic block coding engine, the most memory and bandwidth efficient solution is to set K equal to the block height (e.g. 32 or 64 rows – note that code-blocks need not be square), as discussed carefully in [WG1N1020R]. It is possible to reduce the value of K down to the height of a single sub-block (say 8 or 16 rows), without substantially increasing memory bandwidth, provided the sub-bitplane coding passes use independent probability models ("-Crestart"), as also advocated in [WG1N1201]. This is by no means easy to understand, particularly since a variety of different implementations are enabled. The basic idea is that the coder would process a row of sub-blocks within the current row of code-blocks, within any given pass of the K-line transform, where K is set equal to the block height (it is also possible to set K to any multiple of the sub-block height, but we will not consider that here). This is illustrated roughly in the figure below. All sub-bitplane coding passes must be implemented in parallel (at least conceptually) and at each code-block boundary (in the horizontal direction), the state of these coding engines must be flushed out to memory, to be retrieved later in the next K-line pass, where the next row of sub-blocks in the same set of horizontally adjacent code-blocks is visited. In this process, the significance of the various sub-blocks can be saved to external memory along with the other context and arithmetic coder state information, on horizontal block boundaries, so that the sub-block significance coding process can be executed later, after all sub-block high scans of the code-block have been completed. At this point, the embedded bit-stream for the code-block is also pieced together, from the disjoint pieces corresponding to each coding pass and each sub-block significance scan. In the decoder, the process is reversed.

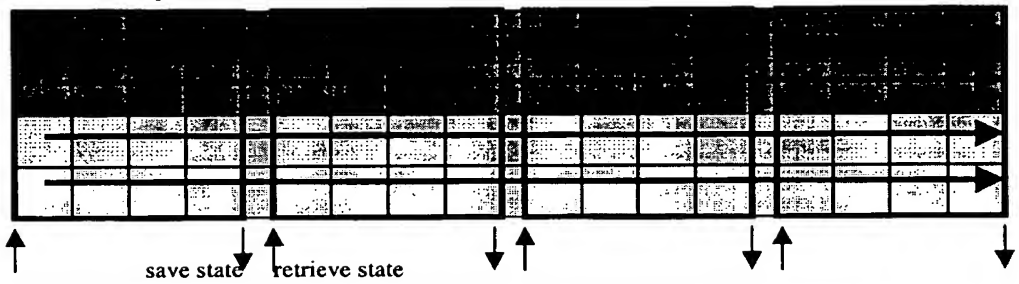

Figure 1 *A row of code-blocks, coded in increments of K lines at a time, where K is the sub-block height. In the figure, there are only 4x4 sub-blocks per code-block. The state of the arithmetic coders for each sub-bitplane pass must all be flushed out and retrieved at the horizontal boundary of each code-block and the significance of each sub-block must also be saved externally, although the latter involves negligible memory and bandwidth.*

Of course, this process of maintaining, flushing and retrieving large amounts of state information on horizontal code-block boundaries is by no means simple and might prove prohibitive in many applications. Nevertheless, the possibility exists to reduce external memory requirements in this way. In practice, it is most likely, that memory savings will be achieved, if desired, by code-blocks which are only a single sub-block high (e.g. 8 or 16 rows) and comparatively wide (e.g. 128 or 256 columns). In this case there is no need to flush the state of many parallel arithmetic coding engines to external memory and the entire code-block bit-stream can be assembled on-chip, which is conceptually and practically very much easier. In this case, one might still choose to use the parallel options (independent probability models for each coding

pass and sub-block causal context formation) even though it has no effect on external memory requirements, in order to reduce on-chip memory requirements. Specifically, with independent probability models for each coding pass and sub-block causal context formation, it is possible to avoid buffering up the code-block samples beyond the width of a sub-block, which is generally much less than the width of a code-block, particularly if very wide blocks are used to compensate for reductions in block height down to that of a single sub-block.

It is unclear whether the parallel options would ever be used in practice solely to reduce external or internal memory requirements. Rather, these options are of use primarily for achieving microscopic parallel encoding and/or decoding for "sample-per-clock" applications in which very high throughput is essential. When used in this way, the memory savings may be seen as an added bonus, particularly when working with wide code-blocks whose height is only that of a single sub-block, since in that case the substantial state information need not be repeatedly stored and retrieved from external memory, as mentioned above. It should also be noted that the memory savings will not be available if the image is to be transposed for one reason or another by decoding and inverse transforming (or forward transforming and encoding) code-blocks in a column by column, rather than row-by-row fashion. They are also not available in the so-called "block-based" applications which have been strongly advocated for digital camera applications. This may be important to some envisaged printing mechanisms.

II-3.3 Performance Loss

Because the size of the sub-block affects the degree of parallelism and the potential for external memory savings, we examine the performance penalty associated with the parallel options here for both 16x16 sub-blocks and 8x8 sub-blocks. In both cases we use the default code-block size of 64x64 samples. To contain the size of this report (in pages, but especially in Mbytes) we present results only for the case in which sub-block causal contexts are combined with the "-Crestart" option. This is the most interesting case since it allows both parallel encoding and parallel decoding. To support parallel decoding we would also need to add markers into the bit-stream to identify the start of each coding pass, but since the cost of this is identical to the cost of adding markers for the "option" coder in VM4 and any other variation on the same theme, there is no need to explicitly report the cost of adding these markers. In fact, provided the sub-block causal contexts are used and the arithmetic coding process is restarted at the beginning of each coding pass, it is always possible to discard and later regenerate the markers without interfering with the rest of the bit-stream. The results for the two different sub-block sizes appear in Table 8 and Table 9.

Table 8 *Comparison of the sequential coding performance with that associated with the parallel options. In both cases, the modifications described in Sections II-1.1, II-1.2, II-1.3, II-1.4 and II-2 are included. Here the default sub-block and code-block sizes of 16x16 and 64x64 respectively are used.*

Lenna (512x512)

Seq bpp	Seq symbols	Seq PSNR	Par bpp (+/- %)	Par Symbols (+/- %)	Par PSNR (+/- dB)
0.06146	0.091	28.18	0.06226 (+1.3)	0.088 (-3.3)	28.09 (-0.09)
0.12497	0.186	31.08	0.12491 (-0.0)	0.179 (-4.0)	30.97 (-0.12)
0.24973	0.374	34.17	0.24924 (-0.2)	0.362 (-3.2)	34.04 (-0.13)
0.49954	0.732	37.32	0.49988 (+0.1)	0.715 (-2.4)	37.21 (-0.10)
0.99829	1.389	40.41	0.99878 (+0.0)	1.367 (-1.6)	40.32 (-0.09)
1.99039	2.522	44.88	1.99792 (+0.4)	2.500 (-0.9)	44.80 (-0.08)

Aerial 2 (2048x2048+A45

Seq bpp	Seq symbols	Seq PSNR	Par bpp (+/- %)	Par Symbols (+/- %)	Par PSNR (+/- dB)
0.06220	0.091	24.63	0.06221 (+0.0)	0.090 (-1.6)	24.60 (-0.03)
0.12487	0.187	26.52	0.12477 (-0.1)	0.184 (-1.2)	26.48 (-0.03)
0.24999	0.353	28.58	0.24984 (-0.1)	0.345 (-2.0)	28.54 (-0.04)
0.49994	0.677	30.63	0.49954 (-0.1)	0.669 (-1.2)	30.60 (-0.04)
0.99913	1.308	33.25	0.99905 (-0.0)	1.297 (-0.8)	33.21 (-0.04)
1.99895	2.449	38.11	1.99660 (-0.1)	2.430 (-0.8)	38.04 (-0.08)

Bike (2048x2560)

Seq bpp	Seq symbols	Seq PSNR	Par bpp (+/- %)	Par Symbols (+/- %)	Par PSNR (+/- dB)
0.06241	0.119	23.81	0.06249 (+0.1)	0.116 (-2.0)	23.75 (-0.05)
0.12451	0.225	26.35	0.12475 (+0.2)	0.220 (-2.3)	26.28 (-0.07)
0.24995	0.431	29.63	0.24999 (+0.0)	0.421 (-2.3)	29.53 (-0.09)
0.49994	0.816	33.52	0.49958 (-0.1)	0.797 (-2.3)	33.40 (-0.12)
0.99980	1.508	38.09	0.99840 (-0.1)	1.477 (-2.0)	37.96 (-0.14)
1.99945	2.743	44.03	1.99778 (-0.1)	2.706 (-1.3)	43.91 (-0.13)

Café (2048x2560)

Seq bpp	Seq symbols	Seq PSNR	Par bpp (+/- %)	Par Symbols (+/- %)	Par PSNR (+/- dB)
0.06240	0.100	19.06	0.06249 (+0.1)	0.098 (-1.5)	19.03 (-0.03)
0.12460	0.208	20.76	0.12491 (+0.2)	0.205 (-1.5)	20.73 (-0.03)
0.24990	0.396	23.14	0.24986 (-0.0)	0.388 (-2.0)	23.07 (-0.07)
0.49960	0.750	26.80	0.49984 (+0.0)	0.737 (-1.8)	26.71 (-0.09)
0.99953	1.444	32.03	0.99934 (-0.0)	1.417 (-1.9)	31.89 (-0.14)
1.99715	2.680	39.07	1.99819 (+0.1)	2.646 (-1.3)	38.92 (-0.14)

Woman (2048x2560)

Seq bpp	Seq symbols	Seq PSNR	Par bpp (+/- %)	Par Symbols (+/- %)	Par PSNR (+/- dB)
0.06222	0.092	25.62	0.06233 (+0.2)	0.091 (-1.3)	25.59 (-0.03)
0.12496	0.188	27.37	0.12426 (-0.6)	0.184 (-1.9)	27.32 (-0.06)
0.24989	0.362	30.01	0.24961 (-0.1)	0.354 (-2.1)	29.94 (-0.08)
0.49974	0.691	33.64	0.49971 (-0.0)	0.679 (-1.7)	33.55 (-0.09)
0.99993	1.332	38.44	0.99979 (-0.0)	1.311 (-1.5)	38.33 (-0.11)
1.99878	2.529	44.04	1.99873 (-0.0)	2.501 (-1.1)	43.94 (-0.10)

Average (Bike, Café, Woman)

Seq bpp	Seq symbols	Seq PSNR	Par bpp (+/- %)	Par Symbols (+/- %)	Par PSNR (+/- dB)
0.06235	0.103	22.83	0.06243 (+0.1)	0.102 (-1.6)	22.79 (-0.04)
0.12469	0.207	24.83	0.12464 (-0.0)	0.203 (-1.9)	24.78 (-0.05)
0.24991	0.396	27.59	0.24982 (-0.0)	0.388 (-2.2)	27.51 (-0.08)
0.49976	0.752	31.32	0.49971 (-0.0)	0.737 (-1.9)	31.22 (-0.10)
0.99975	1.428	36.19	0.99917 (-0.1)	1.402 (-1.8)	36.06 (-0.13)
1.99846	2.651	42.38	1.99823 (-0.0)	2.618 (-1.2)	42.26 (-0.12)

Table 9 *Comparison of the sequential coding performance with that associated with the parallel options. In both cases, the modifications described in Sections II-1.1, II-1.2, II-1.3, II-1.4 and II-2 are included. Here the sub-block and code-block sizes are 8x8 and 64x64 respectively.*

Lenna (512x512)

Seq bpp	Seq symbols	Seq PSNR	Par bpp (+/- %)	Par Symbols (+/- %)	Par PSNR (+/- dB)
0.06155	0.081	28.15	0.06241 (+1.4)	0.079 (-3.4)	28.05 (-0.11)
0.12473	0.165	31.05	0.12485 (+0.1)	0.159 (-3.6)	30.93 (-0.12)
0.24988	0.337	34.16	0.24988	0.323 (-4.1)	34.02 (-0.14)
0.49902	0.663	37.30	0.49982 (+0.2)	0.642 (-3.2)	37.18 (-0.12)
0.99948	1.283	40.42	0.99945 (-0.0)	1.253 (-2.3)	40.30 (-0.12)
1.99454	2.416	44.88	1.99252 (-0.1)	2.375 (-1.7)	44.74 (-0.14)

Aerial 2 (2048x2048+A45

Seq bpp	Seq symbols	Seq PSNR	Par bpp (+/- %)	Par Symbols (+/- %)	Par PSNR (+/- dB)
0.06245	0.083	24.63	0.06232 (-0.2)	0.081 (-1.9)	24.59 (-0.04)
0.12414	0.169	26.49	0.12459 (+0.4)	0.166 (-1.6)	26.45 (-0.04)
0.24999	0.329	28.57	0.24998 (-0.0)	0.321 (-2.4)	28.52 (-0.05)
0.49963	0.630	30.62	0.49955 (-0.0)	0.620 (-1.7)	30.57 (-0.05)
0.99987	1.243	33.25	0.99960 (-0.0)	1.229 (-1.1)	33.19 (-0.06)
1.99770	2.374	38.09	1.99899 (+0.1)	2.354 (-0.8)	38.02 (-0.07)

Bike (2048x2560)

Seq bpp	Seq symbols	Seq PSNR	Par bpp (+/- %)	Par Symbols (+/- %)	Par PSNR (+/- dB)
0.06250	0.103	23.79	0.06249 (-0.0)	0.100 (-3.0)	23.71 (-0.08)
0.12475	0.197	26.33	0.12500 (+0.2)	0.191 (-2.8)	26.24 (-0.08)
0.24950	0.381	29.58	0.24993 (+0.2)	0.369 (-3.0)	29.47 (-0.11)
0.49942	0.731	33.48	0.49937 (-0.0)	0.710 (-2.9)	33.33 (-0.15)
0.99941	1.387	38.06	0.99997 (+0.1)	1.351 (-2.6)	37.91 (-0.15)
1.99934	2.587	44.01	1.99943 (+0.0)	2.540 (-1.8)	43.87 (-0.14)

Café (2048x2560)

Seq bpp	Seq symbols	Seq PSNR	Par bpp (+/- %)	Par Symbols (+/- %)	Par PSNR (+/- dB)
0.06237	0.089	19.04	0.06249 (+0.2)	0.087 (-1.7)	19.00 (-0.03)
0.12459	0.184	20.74	0.12498 (+0.3)	0.180 (-1.8)	20.71 (-0.04)
0.24986	0.358	23.12	0.24921 (-0.3)	0.347 (-2.8)	23.02 (-0.10)
0.49977	0.688	26.78	0.49989 (+0.0)	0.673 (-2.3)	26.66 (-0.12)
0.99915	1.347	32.01	0.99996 (+0.1)	1.313 (-2.5)	31.84 (-0.16)
1.99741	2.548	39.05	1.99800 (+0.0)	2.499 (-1.9)	38.88 (-0.17)

Woman (2048x2560)

Seq bpp	Seq symbols	Seq PSNR	Par bpp (+/- %)	Par Symbols (+/- %)	Par PSNR (+/- dB)
0.06226	0.083	25.62	0.06245 (+0.3)	0.082 (-1.6)	25.58 (-0.04)
0.12461	0.170	27.37	0.12472 (+0.1)	0.166 (-1.9)	27.32 (-0.05)
0.24964	0.332	30.00	0.24992 (+0.1)	0.325 (-2.2)	29.92 (-0.08)
0.49989	0.645	33.64	0.49971 (-0.0)	0.631 (-2.3)	33.52 (-0.11)
0.99935	1.260	38.43	0.99904 (-0.0)	1.235 (-2.0)	38.30 (-0.13)
1.99937	2.433	44.03	1.99830 (-0.1)	2.396 (-1.5)	43.90 (-0.13)

Average (Bike, Café, Woman)

Seq bpp	Seq symbols	Seq PSNR	Par bpp (+/- %)	Par Symbols (+/- %)	Par PSNR (+/- dB)
0.06237	0.092	22.82	0.06248 (+0.2)	0.090 (-2.2)	22.77 (-0.05)
0.12465	0.183	24.81	0.12490 (+0.2)	0.179 (-2.2)	24.76 (-0.06)
0.24967	0.357	27.57	0.24969 (+0.0)	0.347 (-2.7)	27.47 (-0.10)
0.49969	0.688	31.30	0.49966 (-0.0)	0.671 (-2.5)	31.17 (-0.13)
0.99931	1.331	36.17	0.99966 (+0.0)	1.300 (-2.4)	36.02 (-0.15)
1.99871	2.523	42.36	1.99858 (-0.0)	2.478 (-1.8)	42.22 (-0.15)

II-3.4 Comparison of Sub-Block Based vs. Line Based Block Coding

At this point, it is worth comparing two somewhat different approaches to incorporating parallel encoding/decoding capabilities into the block-based entropy coder. The first approach, embodied in the "option" coder within VM4 and described in [WG1N1201], uses a line-based scan pattern within the code-block and line-causal context formation in order to enable parallel decoding. The second approach, described in this document, codes the samples sub-block by sub-block and uses sub-block causal context formation in order to enable parallel decoding. Table 10 and Table 11 compare the compression performance and number of symbols retrieved from the arithmetic coder per image pixel for the two approaches, using sub-block sizes of 16x16 and 8x8, respectively. Based on these results and the peculiar characteristics of the different approaches we may make the following comparative statements:

1) The use of sub-blocks implies a significant reduction in the number of samples which must be visited and coded/decoded during the sub-bitplane coding passes, particularly at lower bit-rates. This in turn implies speed up and/or power reduction in software and hardware implementations.
2) The use of sub-blocks reduces the adaptation overhead of the bitplane coding probability models somewhat, which may have a small positive impact on compression efficiency. Moreover, there is no need here to restrict the context formation to vertically causal contexts in the sub-block based approach. The only modification to contexts is at the lower and right hand sub-block boundaries, which affects only a small proportion of the samples, particularly for larger sub-block sizes. Consequently, loss of compression performance due to the restriction to causal contexts is reduced. The combination of these two effects is evidenced by the results which indicate that the sub-block based approach loses only about half as much (in dB) relative to the original EBCOT coder in VM4 as does the line based approach of the "option" coder. For 8x8 sub-blocks, performance is closer to that of the line based approach, but the reduction in symbol count is also larger -- about 30% at bit-rates of interest.
3) If the parallel options are used to reduce memory consumption, in one of the ways explained in the previous section, then there is no need to buffer K lines of samples across the full code-block width before anything can be coded as is the case with line-based block coding. Instead, it is sufficient to buffer samples for only the width of a single sub-block on-chip, which can represent a substantial saving, particularly when working in the most realistic memory saving mode in which the code-blocks are only a single sub-block high, but relatively wide (say 128 or 256 columns). For this most realistic case, the "option" coder in VM4 provides an alternative "column-by-column" scanning option for precisely this reason, whereas the sub-block based coder described in this document accomplishes a similar effect without introducing a different scanning pattern.
4) A disadvantage of the sub-block based approach is that it restricts the opportunity to realize reduced external memory implementations to the cases in which the number of lines processed at a time is a multiple of the sub-block height. As already mentioned, this may not be a very highly utilized capability for a variety of reasons. Also, a sub-block height of 8 would probably be small enough to achieve most of the useful gain from such schemes without substantially increasing the memory bandwidth.

Table 10 *Comparison of the line-based "option" coder in VM4 (with `-Ccausal') with the sub-block based coder with sub-block causal context formation and the corresponding parallel options. Here the sub-block size is 16x16 and the code-block size is 64x64 in both cases.*

Lenna (512x512)

Line bpp	Line symbols	Line PSNR	Sub bpp (+/-%)	Sub Symbols (+/-%)	Sub PSNR (+/-dB)
0.06235	0.124	28.04	0.06226 (-0.1)	0.088 (-28.6)	28.09 (+0.05)
0.12439	0.245	30.88	0.12491 (+0.4)	0.179 (-27.0)	30.97 (+0.08)
0.24976	0.493	33.95	0.24924 (-0.2)	0.362 (-26.7)	34.04 (+0.09)
0.49957	0.964	37.10	0.49988 (+0.1)	0.715 (-25.8)	37.21 (+0.12)
0.99841	1.706	40.21	0.99878 (+0.0)	1.367 (-19.9)	40.32 (+0.10)
1.98062	2.851	44.63	1.99792 (+0.9)	2.500 (-12.3)	44.80 (+0.17)

Aerial 2 (2048x2048+A45

Line bpp	Line symbols	Line PSNR	Sub bpp (+/-%)	Sub Symbols (+/-%)	Sub PSNR (+/-dB)
0.06206	0.114	24.59	0.06221 (+0.3)	0.090 (-21.1)	24.60 (+0.01)
0.12421	0.236	26.47	0.12477 (+0.5)	0.184 (-21.9)	26.48 (+0.02)
0.24982	0.419	28.52	0.24984 (+0.0)	0.345 (-17.6)	28.54 (+0.02)
0.49974	0.801	30.58	0.49954 (-0.0)	0.669 (-16.5)	30.60 (+0.02)
0.99978	1.485	33.20	0.99905 (-0.1)	1.297 (-12.6)	33.21 (+0.01)
1.99797	2.653	38.03	1.99660 (-0.1)	2.430 (-8.4)	38.04 (+0.01)

Bike (2048x2560)

Line bpp	Line symbols	Line PSNR	Sub bpp (+/-%)	Sub Symbols (+/-%)	Sub PSNR (+/-dB)
0.06248	0.174	23.71	0.06249 (+0.0)	0.116 (-33.1)	23.75 (+0.05)
0.12494	0.323	26.23	0.12475 (-0.2)	0.220 (-31.9)	26.28 (+0.05)
0.24988	0.602	29.43	0.24999 (+0.0)	0.421 (-30.0)	29.53 (+0.11)
0.49951	1.059	33.25	0.49958 (+0.0)	0.797 (-24.7)	33.40 (+0.15)
0.99977	1.816	37.81	0.99840 (-0.1)	1.477 (-18.6)	37.96 (+0.15)
1.99967	3.093	43.79	1.99778 (-0.1)	2.706 (-12.5)	43.91 (+0.12)

Café (2048x2560)

Line bpp	Line symbols	Line PSNR	Sub bpp (+/-%)	Sub Symbols (+/-%)	Sub PSNR (+/-dB)
0.06249	0.121	19.00	0.06249 (-0.0)	0.098 (-18.9)	19.03 (+0.02)
0.12471	0.256	20.72	0.12491 (+0.2)	0.205 (-19.9)	20.73 (+0.01)
0.24978	0.478	23.04	0.24986 (+0.0)	0.388 (-18.7)	23.07 (+0.02)
0.49980	0.887	26.61	0.49984 (+0.0)	0.737 (-16.9)	26.71 (+0.10)
0.99942	1.636	31.74	0.99934 (-0.0)	1.417 (-13.4)	31.89 (+0.16)
1.99896	2.917	38.78	1.99819 (-0.0)	2.646 (-9.3)	38.92 (+0.14)

Woman (2048x2560)

Line bpp	Line symbols	Line PSNR	Sub bpp (+/-%)	Sub Symbols (+/-%)	Sub PSNR (+/-dB)
0.06224	0.128	25.54	0.06233 (+0.1)	0.091 (-29.0)	25.59 (+0.05)
0.12478	0.265	27.28	0.12426 (-0.4)	0.184 (-30.5)	27.32 (+0.03)
0.24995	0.471	29.85	0.24961 (-0.1)	0.354 (-24.9)	29.94 (+0.08)
0.49986	0.858	33.43	0.49971 (-0.0)	0.679 (-20.9)	33.55 (+0.12)
0.99966	1.573	38.18	0.99979 (+0.0)	1.311 (-16.7)	38.33 (+0.15)
1.99897	2.817	43.81	1.99873 (-0.0)	2.501 (-11.2)	43.94 (+0.12)

Average (Bike, Café, Woman)

Line bpp	Line symbols	Line PSNR	Sub bpp (+/-%)	Sub Symbols (+/-%)	Sub PSNR (+/-dB)
0.06240	0.141	22.75	0.06243 (+0.1)	0.102 (-27.8)	22.79 (+0.04)
0.12481	0.281	24.75	0.12464 (-0.1)	0.203 (-27.8)	24.78 (+0.03)
0.24987	0.517	27.44	0.24982 (-0.0)	0.388 (-25.0)	27.51 (+0.07)
0.49972	0.935	31.10	0.49971 (-0.0)	0.737 (-21.1)	31.22 (+0.12)
0.99962	1.675	35.91	0.99917 (-0.0)	1.402 (-16.3)	36.06 (+0.15)
1.99920	2.942	42.13	1.99823 (-0.0)	2.618 (-11.0)	42.26 (+0.13)

Table 11 *Comparison of the line-based "option" coder in VM4 (with `-Ccausal') with the sub-block based coder with sub-block causal context formation and the corresponding parallel options. Here the sub-block size is 8x8 and the code-block size is 64x64 in both cases.*

Lenna (512x512)

Line bpp	Line symbols	Line PSNR	Sub bpp (+/-%)	Sub Symbols (+/-%)	Sub PSNR (+/-dB)
0.06235	0.124	28.04	0.06241 (+0.1)	0.079 (-36.5)	28.05 (+0.00)
0.12439	0.245	30.88	0.12485 (+0.4)	0.159 (-35.0)	30.93 (+0.05)
0.24976	0.493	33.95	0.24988 (+0.0)	0.323 (-34.5)	34.02 (+0.07)
0.49957	0.964	37.10	0.49982 (+0.0)	0.642 (-33.4)	37.18 (+0.08)
0.99841	1.706	40.21	0.99945 (+0.1)	1.253 (-26.5)	40.30 (+0.08)
1.98062	2.851	44.63	1.99252 (+0.6)	2.375 (-16.7)	44.74 (+0.11)

Aerial 2 (2048x2048+A45

Line bpp	Line symbols	Line PSNR	Sub bpp (+/-%)	Sub Symbols (+/-%)	Sub PSNR (+/-dB)
0.06206	0.114	24.59	0.06232 (+0.4)	0.081 (-28.3)	24.59 (+0.00)
0.12421	0.236	26.47	0.12459 (+0.3)	0.166 (-29.7)	26.45 (-0.02)
0.24982	0.419	28.52	0.24998 (+0.1)	0.321 (-23.5)	28.52 (+0.00)
0.49974	0.801	30.58	0.49955 (-0.0)	0.620 (-22.7)	30.57 (-0.00)
0.99978	1.485	33.20	0.99960 (-0.0)	1.229 (-17.2)	33.19 (-0.01)
1.99797	2.653	38.03	1.99899 (+0.1)	2.354 (-11.3)	38.02 (-0.01)

Bike (2048x2560)

Line bpp	Line symbols	Line PSNR	Sub bpp (+/-%)	Sub Symbols (+/-%)	Sub PSNR (+/-dB)
0.06248	0.174	23.71	0.06249 (+0.0)	0.100 (-42.5)	23.71 (+0.01)
0.12494	0.323	26.23	0.12500 (+0.0)	0.191 (-40.7)	26.24 (+0.01)
0.24988	0.602	29.43	0.24993 (+0.0)	0.369 (-38.6)	29.47 (+0.04)
0.49951	1.059	33.25	0.49937 (-0.0)	0.710 (-33.0)	33.33 (+0.08)
0.99977	1.816	37.81	0.99997 (+0.0)	1.351 (-25.6)	37.91 (+0.10)
1.99967	3.093	43.79	1.99943 (-0.0)	2.540 (-17.9)	43.87 (+0.08)

Café (2048x2560)

Line bpp	Line symbols	Line PSNR	Sub bpp (+/-%)	Sub Symbols (+/-%)	Sub PSNR (+/-dB)
0.06249	0.121	19.00	0.06249	0.087 (-28.1)	19.00 (+0.00)
0.12471	0.256	20.72	0.12498 (+0.2)	0.180 (-29.6)	20.71 (-0.01)
0.24978	0.478	23.04	0.24921 (-0.2)	0.347 (-27.2)	23.02 (-0.03)
0.49980	0.887	26.61	0.49989 (+0.0)	0.673 (-24.1)	26.66 (+0.05)
0.99942	1.636	31.74	0.99996 (+0.1)	1.313 (-19.7)	31.84 (+0.11)
1.99896	2.917	38.78	1.99800 (-0.0)	2.499 (-14.3)	38.88 (+0.10)

Woman (2048x2560)

Line bpp	Line symbols	Line PSNR	Sub bpp (+/-%)	Sub Symbols (+/-%)	Sub PSNR (+/-dB)
0.06224	0.128	25.54	0.06245 (+0.3)	0.082 (-35.7)	25.58 (+0.05)
0.12478	0.265	27.28	0.12472 (-0.0)	0.166 (-37.3)	27.32 (+0.04)
0.24995	0.471	29.85	0.24992 (-0.0)	0.325 (-31.0)	29.92 (+0.07)
0.49986	0.858	33.43	0.49971 (-0.0)	0.631 (-26.5)	33.52 (+0.09)
0.99966	1.573	38.18	0.99904 (-0.1)	1.235 (-21.5)	38.30 (+0.11)
1.99897	2.817	43.81	1.99830 (-0.0)	2.396 (-15.0)	43.90 (+0.09)

Average (Bike, Café, Woman)

Line bpp	Line symbols	Line PSNR	Sub bpp (+/-%)	Sub Symbols (+/-%)	Sub PSNR (+/-dB)
0.06240	0.141	22.75	0.06248 (+0.1)	0.090 (-36.3)	22.77 (+0.02)
0.12481	0.281	24.75	0.12490 (+0.1)	0.179 (-36.2)	24.76 (+0.01)
0.24987	0.517	27.44	0.24969 (-0.1)	0.347 (-32.8)	27.47 (+0.03)
0.49972	0.935	31.10	0.49966 (-0.0)	0.671 (-28.2)	31.17 (+0.08)
0.99962	1.675	35.91	0.99966 (+0.0)	1.300 (-22.4)	36.02 (+0.11)
1.99920	2.942	42.13	1.99858 (-0.0)	2.478 (-15.8)	42.22 (+0.09)

II-4 A Lazy Coding Mode

In Section II-3, we concentrated on achieving properties such as parallel encoding/decoding while minimizing the average number of symbols to be coded/decoded. In this section we investigate one final modification, which again we propose as an option, which both reduces the average number of arithmetically coded symbols and also reduces the maximum number of coding passes in which symbols might need to be arithmetically coded, which can be of advantage in simplifying hardware implementations. The idea is very simple and is connected with techniques used in Ricoh's "CREW" algorithm. Specifically, we begin by observing that the probability models for many coding contexts attain distributions close to uniform in the least significant bit-planes. It is a waste of effort to use the arithmetic coding engine to code these binary symbols; instead, we prefer to send them as raw binary digits. Although it is difficult to interleave raw binary digits into an arithmetically coded bit-stream, it is possible to bypass the arithmetic coding engine altogether for an entire sub-bitplane coding pass provided the arithmetic coder is terminated at the end of the previous coding pass (using the Elias termination which allows for unique decoding with arbitrary bit-stream suffices) and restarted at the beginning of the next pass which requires arithmetic coding. This is a subset of the behaviour offered by the "-Crestart" option and for our experiments so the software implementation supports the behaviour both with and without the costly reinitialization of coding context states which goes with the "-Crestart" mode. For our experiments, however, we prefer to use this "lazy" mode only in conjunction with "-Crestart" and sub-block causal contexts for reasons which will be explained shortly. To be specific, in the proposed, optional modification, all of the binary symbols generated in the "significance propagation" and "magnitude refinement" coding passes (see Section II-1.2) representing bits in bit-planes $p < p_0 - K$ are written directly into the bit-stream as raw binary digits, entirely bypassing the arithmetic coder, where p_0 denotes the most significant bit-plane in which any sample in the relevant code-block becomes significant and K is a parameter which we suggest should be set to $K = 3$. It should be pointed out that an additional modification to the Elias termination procedure was required in order to ensure that this "lazy" mode could be used in conjunction with the MQ coder. Specifically, since the MQ coder is byte-oriented, with a bit-stuffing rather than carry propagation policy for dealing with carry generation at the encoder, the arbitrary bit-stream suffices which can be generated by the emission of raw uncoded bits can generate illegal bit-stream for a previous MQ-coded pass. To avoid this difficulty, we modified the Elias termination implementation to allow for truly arbitrary suffices; the details of this modification are not warranted by the scope of this document, but they are adequately described by comments in the source code.

One advantage of the modification is that it substantially reduces the number of symbols which must be arithmetically coded at high bit-rates. Also, since we usually encode all code-blocks in an image at a high rate before truncating down to a final target bit-rate, this scheme substantially reduces the number of symbols which must typically be encoded and hence reduces the encoding time. We find, for example, that CPU times for reversible compression are typically reduced by 30%. On the other hand, the modification has relatively little effect on compression performance.

The second advantage of the modification is that it substantially reduces the maximum number of coding passes in which arithmetic coding might need to be used. Without the modification, the maximum number of coding passes for any given code-block is $3P_{max} - 2$ where P_{max} is the maximum number of bit-planes in any given subband and might be on the order of 12 for the lower frequency subbands. On the other hand, with the modification, the maximum number of coding passes for any given code-block is $P_{max} + 2K = P_{max} + 6$. In applications where we intend to use microscopic parallelism to achieve "sample-per-clock" throughput, this means a substantial reduction in the number of parallel arithmetic coding engines which must be included on the chip. Due to the importance of the combination of this mode with parallel encoding/decoding, we provide results in Table 1 for the combination of this so-called "lazy" option with the "-Crestart" option and sub-block causal coding contexts, comparing the performance with the use of sub-block causal coding contexts and "-Crestart" alone.

Table 12 *Comparison of the "lazy coding" option in combination with the parallel options with the parallel*

options alone, for a sub-block size of 16x16 and a code-block size of 64x64.

Lenna (512x512)

Par bpp	Par symbols	Par PSNR	Lazy bpp (+/-%)	Lazy Symbols (+/-%)	Lazy PSNR (+/-dB)
0.06226	0.088	28.09	0.06223 (-0.0)	0.085 (-3.6)	28.09
0.12491	0.179	30.97	0.12491	0.167 (-7.0)	30.96 (-0.00)
0.24924	0.362	34.04	0.24970 (+0.2)	0.310 (-14.2)	33.99 (-0.05)
0.49988	0.715	37.21	0.49973 (-0.0)	0.580 (-18.8)	37.14 (-0.07)
0.99878	1.367	40.32	0.99966 (+0.1)	1.041 (-23.9)	40.28 (-0.04)
1.99792	2.500	44.80	1.98401 (-0.7)	1.731 (-30.8)	44.75 (-0.05)

Aerial 2 (2048x2048+A45

Par bpp	Par symbols	Par PSNR	Lazy bpp (+/-%)	Lazy Symbols (+/-%)	Lazy PSNR (+/-dB)
0.06221	0.090	24.60	0.06203 (-0.3)	0.080 (-10.7)	24.59 (-0.00)
0.12477	0.184	26.48	0.12472 (-0.0)	0.164 (-11.2)	26.49 (+0.00)
0.24984	0.345	28.54	0.24991 (+0.0)	0.304 (-12.0)	28.54 (+0.01)
0.49954	0.669	30.60	0.49966 (+0.0)	0.585 (-12.5)	30.60 (+0.00)
0.99905	1.297	33.21	0.99906 (+0.0)	1.089 (-16.0)	33.23 (+0.01)
1.99660	2.430	38.04	1.99556 (-0.1)	2.008 (-17.4)	38.08 (+0.04)

Bike (2048x2560)

Par bpp	Par symbols	Par PSNR	Lazy bpp (+/-%)	Lazy Symbols (+/-%)	Lazy PSNR (+/-dB)
0.06249	0.116	23.75	0.06248 (-0.0)	0.111 (-4.3)	23.74 (-0.01)
0.12475	0.220	26.28	0.12486 (+0.1)	0.204 (-7.3)	26.24 (-0.04)
0.24999	0.421	29.53	0.24987 (-0.0)	0.377 (-10.5)	29.46 (-0.07)
0.49958	0.797	33.40	0.49997 (+0.1)	0.680 (-14.7)	33.28 (-0.12)
0.99840	1.477	37.96	0.99921 (+0.1)	1.166 (-21.1)	37.80 (-0.16)
1.99778	2.706	43.91	1.99976 (+0.1)	1.844 (-31.9)	43.75 (-0.16)

Café (2048x2560)

Par bpp	Par symbols	Par PSNR	Lazy bpp (+/-%)	Lazy Symbols (+/-%)	Lazy PSNR (+/-dB)
0.06249	0.098	19.03	0.06237 (-0.2)	0.096 (-2.9)	19.02 (-0.00)
0.12491	0.205	20.73	0.12490 (-0.0)	0.200 (-2.8)	20.73 (+0.00)
0.24986	0.388	23.07	0.24999 (+0.1)	0.378 (-2.8)	23.07 (+0.00)
0.49984	0.737	26.71	0.49993 (+0.0)	0.694 (-5.9)	26.70 (-0.01)
0.99934	1.417	31.89	0.99976 (+0.0)	1.255 (-11.4)	31.85 (-0.04)
1.99819	2.646	38.92	1.99956 (+0.1)	1.931 (-27.0)	38.85 (-0.07)

Woman (2048x2560)

Par bpp	Par symbols	Par PSNR	Lazy bpp (+/-%)	Lazy Symbols (+/-%)	Lazy PSNR (+/-dB)
0.06233	0.091	25.59	0.06243 (+0.2)	0.087 (-4.0)	25.59 (-0.00)
0.12426	0.184	27.32	0.12459 (+0.3)	0.180 (-2.6)	27.32 (+0.00)
0.24961	0.354	29.94	0.25000 (+0.2)	0.340 (-3.9)	29.93 (-0.00)
0.49971	0.679	33.55	0.49963 (-0.0)	0.627 (-7.6)	33.54 (-0.01)
0.99979	1.311	38.33	0.99989 (+0.0)	1.089 (-17.0)	38.31 (-0.03)
1.99873	2.501	43.94	1.99651 (-0.1)	1.772 (-29.1)	43.91 (-0.02)

Average (Bike, Café, Woman)

Par bpp	Par symbols	Par PSNR	Lazy bpp (+/-%)	Lazy Symbols (+/-%)	Lazy PSNR (+/-dB)
0.06243	0.102	22.79	0.06243 (-0.0)	0.098 (-3.8)	22.78 (-0.01)
0.12464	0.203	24.78	0.12478 (+0.1)	0.194 (-4.3)	24.76 (-0.01)
0.24982	0.388	27.51	0.24995 (+0.1)	0.365 (-5.9)	27.49 (-0.02)
0.49971	0.737	31.22	0.49984 (+0.0)	0.667 (-9.6)	31.17 (-0.05)
0.99917	1.402	36.06	0.99962 (+0.0)	1.170 (-16.5)	35.99 (-0.08)
1.99823	2.618	42.26	1.99861 (+0.0)	1.849 (-29.4)	42.17 (-0.08)

To conclude this section, it is worthwhile comparing the lossless performance associated with the various different modifications and options which have been advanced. Table 2 provides this comparison in terms of lossless bit-rate and total number of arithmetically coded symbols per sample, for five different algorithms.

Table 13 *Comparison of lossless coding performance for five different algorithms (mode variations) using 64x64 code-blocks and 16x16 sub-blocks where applicable with the 5/3 default reversible Wavelet kernel. The first pair of columns refer to VM4; the second pair refer to the coder obtained by applying the modifications described in Sections II-1 and II-2; the third pair of columns is obtained by adding the parallel options ("-Crestart" and sub-block causal context formation); the fourth pair of columns are obtained by adding the "lazy" coding option; and the last pair of columns are obtained using the "option" coder from VM4 with "-Ccausal".*

VM4 bpp	VM4 Syms	Mod bpp	Mod Syms	Par bpp	Par Syms	Lazy bpp	Lazy Syms	Option bpp	Option Syms
Lenna (512x512)									
4.3024	4:87	4.3087	4:80	4.3344	4:80	4.3078	2:43	4.3553	5:1
Aerial2 (2048x2048)									
5.4502	5:88	5.4495	5:81	5.4700	5:81	5.3874	3:20	5.4793	6:0
Bike (2048x2560)									
4.5339	5:36	4.5373	5:27	4.5677	5:26	4.5667	2:37	4.5993	5:6
Cafe (2048x2560)									
5.3557	6:17	5.3598	6:07	5.3967	6:06	5.3527	2:59	5.4314	6:3
Woman (2048x2560)									
4.5158	5:11	4.5171	5:04	4.5429	5:04	4.5171	2:67	4.5758	5:3
Average (Bike, Café, Woman)									
4.8018	5:55	4.8047	5:46	4.8358	5:45	4.8122	2:54	4.8688	5:8

Some of the interesting points to observe are as follows:

- The "lazy" coding mode requires far fewer (usually less than half as many) symbols to be coded than any of the other modes.
- The "lazy" coding mode generates a lower compressed bit-rate than that obtained with the same parallel options but the lazy mode turned off. Since the lazy mode affects only the significance propagation and magnitude refinement coding passes all of whose coding contexts are initialized to the MQ coder's standard initial state at the beginning of the "learning curve", this result indicates that the relevant distributions must be so close to uniform that emitting raw binary digits is more efficient than letting the arithmetic coder learn this uniform distribution.
- The modifications to the original EBCOT algorithm described in Sections II-1 and II-2 have negligible effect on lossless performance.
- The "option" coder in VM4 codes slightly more symbols and compresses slightly less efficiently than all the other modes.

III CPU TIMES FOR VARIOUS MODES OF INTEREST

In this section we provide CPU decoding times for each different bit-rate and each of the JPEG2000 images we have been considering for a number of algorithm modes. The CPU times are all obtained using a 400 MHz Pentium Pro (perhaps the most appropriate target platform for initial consumer applications of JPEG2000). To reduce the substantial jitter in CPU timing results, we added a loop into the implementation of "decode_block" which iterates five (5) times through the block decoding process between calls to the standard ANSI "C" library function, "clock()", which is used to measure the CPU time. The same modification was made to the VM4.1 source code for both the "option" coder and the original EBCOT variation, so as to ensure that the results can be reliably compared. In this section we work with the latest version of the VM4.1 source code for comparisons with existing entropy coder options.

The most interesting results are distilled in the four tables which follow. The captions for these tables are self explanatory and so we simply summarize here some of the conclusions which may be drawn from these results:

- The proposed modifications to the EBCOT coder in VM4.1 yield improvements in software execution speed on the order of 15% with negligible loss in compression performance.
- If 8x8 sub-blocks are used instead of 16x16, performance degrades very little while software execution speed increases by about 6 or 7% (more at lower bit-rates).
- With 8x8 sub-blocks the modified EBCOT coder runs about 15% faster than the "option" coder in VM4.1, with slight improvements in compression performance, when operated in comparable modes.
- The "lazy" coding option yields another 10% improvement in software execution speed at moderate bit-rates, while introducing a slight loss in compression efficiency. At least in the case where the parallel options are enabled, the "lazy" coding option to the modified EBCOT coder yields almost identical compression performance to the "option" coder in VM4.1 with a speed-up of about 25%. The speed-up factor increases rapidly at very high bit-rates without any further loss in compression efficiency.

Table 14 *Comparison of CPU times (and PSNR) between VM4.1 and the EBCOT coder modified as described in Sections II and II-2, with 64x64 code-blocks and 16x16 sub-blocks in both cases.*

Aerial 2 (2048x2048+A45

VM4.1 bpp	VM4.1 s/Mpel	VM4.1 PSNR	Mod bpp (+/-%)	Mod s/Mpel (speedup%)	Mod PSNR (+/-dB)
0.06237	0.030	24.66	0.06220 (-0.3)	0.030	24.63 (-0.03)
0.12473	0.060	26.53	0.12487 (+0.1)	0.059 (+1.5)	26.52 (-0.01)
0.24984	0.124	28.58	0.24999 (+0.1)	0.108 (+14.6)	28.58 (+0.00)
0.49924	0.237	30.63	0.49994 (+0.1)	0.209 (+13.5)	30.63 (+0.00)
0.99959	0.432	33.27	0.99913 (-0.0)	0.371 (+16.4)	33.25 (-0.01)
1.99802	0.834	38.11	1.99895 (+0.0)	0.730 (+14.2)	38.11 (+0.00)

Bike (2048x2560)

VM4.1 bpp	VM4.1 s/Mpel	VM4.1 PSNR	Mod bpp (+/-%)	Mod s/Mpel (speedup%)	Mod PSNR (+/-dB)
0.06249	0.039	23.83	0.06241 (-0.1)	0.040 (-1.8)	23.81 (-0.02)
0.12475	0.074	26.41	0.12451 (-0.2)	0.070 (+6.6)	26.35 (-0.06)
0.24977	0.147	29.66	0.24995 (+0.1)	0.128 (+14.2)	29.63 (-0.03)
0.49932	0.287	33.54	0.49994 (+0.1)	0.249 (+15.1)	33.52 (-0.02)
0.99989	0.520	38.12	0.99980 (-0.0)	0.455 (+14.2)	38.09 (-0.02)
1.99581	0.955	44.03	1.99945 (+0.2)	0.840 (+13.6)	44.03 (-0.00)

Café (2048x2560)

VM4.1 bpp	VM4.1 s/Mpel	VM4.1 PSNR	Mod bpp (+/-%)	Mod s/Mpel (speedup%)	Mod PSNR (+/-dB)
0.06238	0.032	19.07	0.06240 (+0.0)	0.028 (+15.2)	19.06 (-0.01)
0.12496	0.067	20.81	0.12460 (-0.3)	0.057 (+17.2)	20.76 (-0.05)
0.24997	0.132	23.16	0.24990 (-0.0)	0.112 (+18.0)	23.14 (-0.02)
0.49949	0.253	26.83	0.49960 (+0.0)	0.216 (+17.2)	26.80 (-0.03)
0.99910	0.495	32.06	0.99953 (+0.0)	0.437 (+13.3)	32.03 (-0.03)
1.99875	0.950	39.12	1.99715 (-0.1)	0.850 (+11.8)	39.07 (-0.05)

Woman (2048x2560)

VM4.1 bpp	VM4.1 s/Mpel	VM4.1 PSNR	Mod bpp (+/-%)	Mod s/Mpel (speedup%)	Mod PSNR (+/-dB)
0.06214	0.031	25.63	0.06222 (+0.1)	0.033 (-4.6)	25.62 (-0.01)
0.12482	0.061	27.39	0.12496 (+0.1)	0.058 (+5.4)	27.37 (-0.02)
0.24975	0.119	30.02	0.24989 (+0.1)	0.109 (+9.6)	30.01 (-0.01)
0.49956	0.245	33.66	0.49974 (+0.0)	0.210 (+17.0)	33.64 (-0.01)
0.99989	0.467	38.45	0.99993 (+0.0)	0.413 (+13.2)	38.44 (-0.01)
1.99877	0.889	44.05	1.99878 (+0.0)	0.782 (+13.7)	44.04 (-0.01)

Average (Bike, Café, Woman)

VM4.1 bpp	VM4.1 s/Mpel	VM4.1 PSNR	Mod bpp (+/-%)	Mod s/Mpel (speedup%)	Mod PSNR (+/-dB)
0.06234	0.034	22.84	0.06235 (+0.0)	0.034 (+2.0)	22.83 (-0.01)
0.12484	0.067	24.87	0.12469 (-0.1)	0.062 (+9.5)	24.83 (-0.04)
0.24983	0.133	27.61	0.24991 (+0.0)	0.116 (+14.0)	27.59 (-0.02)
0.49946	0.262	31.34	0.49976 (+0.1)	0.225 (+16.3)	31.32 (-0.02)
0.99963	0.494	36.21	0.99975 (+0.0)	0.435 (+13.6)	36.19 (-0.02)
1.99778	0.931	42.40	1.99846 (+0.0)	0.824 (+13.0)	42.38 (-0.02)

Table 15 *Comparison of CPU times (and PSNR) for the modified EBCOT coder described in Sections II and II-2,*

using 16x16 and 8x8 sub-blocks, respectively, with 64x64 code-blocks in both cases.

Aerial 2 (2048x2048+A45

16x16 bpp	16x16 s/Mpel	16x16 PSNR	8x8 bpp (+/- %)	8x8 s/Mpel (speedup %)	8x8 PSNR (+/- dB)
0.06220	0.030	24.63	0.06245 (+0.4)	0.023 (+26.5)	24.63 (+0.00)
0.12487	0.059	26.52	0.12414 (-0.6)	0.049 (+19.5)	26.49 (-0.02)
0.24999	0.108	28.58	0.24999 (-0.0)	0.099 (+9.1)	28.57 (-0.01)
0.49994	0.209	30.63	0.49963 (-0.1)	0.207 (+0.9)	30.62 (-0.01)
0.99913	0.371	33.25	0.99987 (+0.1)	0.359 (+3.4)	33.25 (-0.01)
1.99895	0.730	38.11	1.99770 (-0.1)	0.710 (+2.9)	38.09 (-0.02)

Bike (2048x2560)

16x16 bpp	16x16 s/Mpel	16x16 PSNR	8x8 bpp (+/- %)	8x8 s/Mpel (speedup %)	8x8 PSNR (+/- dB)
0.06241	0.040	23.81	0.06250 (+0.1)	0.028 (+45.6)	23.79 (-0.02)
0.12451	0.070	26.35	0.12475 (+0.2)	0.060 (+16.6)	26.33 (-0.02)
0.24995	0.128	29.63	0.24950 (-0.2)	0.121 (+6.5)	29.58 (-0.05)
0.49994	0.249	33.52	0.49942 (-0.1)	0.226 (+10.3)	33.48 (-0.04)
0.99980	0.455	38.09	0.99941 (-0.0)	0.429 (+6.2)	38.06 (-0.03)
1.99945	0.840	44.03	1.99934 (-0.0)	0.813 (+3.4)	44.01 (-0.02)

Café (2048x2560)

16x16 bpp	16x16 s/Mpel	16x16 PSNR	8x8 bpp (+/- %)	8x8 s/Mpel (speedup %)	8x8 PSNR (+/- dB)
0.06240	0.028	19.06	0.06237 (-0.0)	0.023 (+21.7)	19.04 (-0.02)
0.12460	0.057	20.76	0.12459 (-0.0)	0.055 (+5.0)	20.74 (-0.01)
0.24990	0.112	23.14	0.24986 (-0.0)	0.105 (+6.2)	23.12 (-0.02)
0.49960	0.216	26.80	0.49977 (+0.0)	0.204 (+5.9)	26.78 (-0.02)
0.99953	0.437	32.03	0.99915 (-0.0)	0.411 (+6.4)	32.01 (-0.02)
1.99715	0.850	39.07	1.99741 (+0.0)	0.813 (+4.5)	39.05 (-0.01)

Woman (2048x2560)

16x16 bpp	16x16 s/Mpel	16x16 PSNR	8x8 bpp (+/- %)	8x8 s/Mpel (speedup %)	8x8 PSNR (+/- dB)
0.06222	0.033	25.62	0.06226 (+0.1)	0.026 (+28.5)	25.62 (+0.00)
0.12496	0.058	27.37	0.12461 (-0.3)	0.050 (+14.4)	27.37 (-0.01)
0.24989	0.109	30.01	0.24964 (-0.1)	0.107 (+1.5)	30.00 (-0.01)
0.49974	0.210	33.64	0.49989 (+0.0)	0.199 (+5.2)	33.64 (-0.01)
0.99993	0.413	38.44	0.99935 (-0.1)	0.400 (+3.2)	38.43 (-0.01)
1.99878	0.782	44.04	1.99937 (+0.0)	0.770 (+1.5)	44.03 (-0.01)

Average (Bike, Café, Woman)

16x16 bpp	16x16 s/Mpel	16x16 PSNR	8x8 bpp (+/-%)	8x8 s/Mpel (speedup %)	8x8 PSNR (+/- dB)
0.06235	0.034	22.83	0.06237 (+0.0)	0.025 (+32.6)	22.82 (-0.01)
0.12469	0.062	24.83	0.12465 (-0.0)	0.055 (+12.1)	24.81 (-0.01)
0.24991	0.116	27.59	0.24967 (-0.1)	0.111 (+4.8)	27.57 (-0.03)
0.49976	0.225	31.32	0.49969 (-0.0)	0.210 (+7.3)	31.30 (-0.02)
0.99975	0.435	36.19	0.99931 (-0.0)	0.413 (+5.3)	36.17 (-0.02)
1.99846	0.824	42.38	1.99871 (+0.0)	0.798 (+3.2)	42.36 (-0.02)

Table 16 *Comparison of CPU times (and PSNR) between the VM4.1 "option" coder with "-Ccausal" and the modified EBCOT coder with comparable parallel options, "-Crestart" and sub-block causal contexts. The sub-block size in this case was set to 8x8. Code-blocks are 64x64 in both cases.*

Aerial 2 (2048x2048+A45

option bpp	option s/Mpel	option PSNR	para bpp (+/- %)	para s/Mpel (speedup %)	para PSNR (+/- dB)
0.06192	0.036	24.59	0.06222 (+0.5)	0.023 (+53.3)	24.58 (-0.00)
0.12472	0.064	26.48	0.12425 (-0.4)	0.054 (+16.7)	26.46 (-0.02)
0.24998	0.112	28.52	0.24976 (-0.1)	0.104 (+7.3)	28.52 (-0.00)
0.49941	0.226	30.58	0.49947 (+0.0)	0.197 (+15.0)	30.58 (-0.00)
0.99904	0.417	33.20	0.99937 (+0.0)	0.349 (+19.4)	33.20 (-0.00)
1.99399	0.806	38.03	1.99833 (+0.2)	0.702 (+14.7)	38.03 (+0.00)

Bike (2048x2560)

option bpp	option s/Mpel	option PSNR	para bpp (+/- %)	para s/Mpel (speedup %)	para PSNR (+/- dB)
0.06239	0.039	23.71	0.06242 (+0.0)	0.028 (+42.9)	23.72 (+0.01)
0.12483	0.065	26.24	0.12468 (-0.1)	0.059 (+10.4)	26.24 (+0.00)
0.24989	0.139	29.42	0.24989 (+0.0)	0.124 (+12.0)	29.48 (+0.05)
0.49975	0.261	33.25	0.49945 (-0.1)	0.221 (+17.8)	33.34 (+0.09)
0.99978	0.490	37.82	0.99999 (+0.0)	0.419 (+17.0)	37.92 (+0.10)
1.99944	0.903	43.80	1.99732 (-0.1)	0.805 (+12.2)	43.87 (+0.07)

Café (2048x2560)

option bpp	option s/Mpel	option PSNR	para bpp (+/- %)	para s/Mpel (speedup %)	para PSNR (+/- dB)
0.06223	0.034	19.00	0.06237 (+0.2)	0.026 (+34.3)	19.00 (+0.00)
0.12486	0.065	20.73	0.12500 (+0.1)	0.058 (+11.2)	20.71 (-0.01)
0.24983	0.131	23.05	0.24964 (-0.1)	0.103 (+27.4)	23.04 (-0.01)
0.49990	0.229	26.61	0.49997 (+0.0)	0.203 (+12.7)	26.67 (+0.06)
0.99963	0.465	31.74	0.99959 (-0.0)	0.401 (+15.9)	31.85 (+0.11)
1.99779	0.889	38.78	1.99928 (+0.1)	0.802 (+10.9)	38.90 (+0.11)

Woman (2048x2560)

option bpp	option s/Mpel	option PSNR	para bpp (+/- %)	para s/Mpel (speedup %)	para PSNR (+/- dB)
0.06218	0.031	25.53	0.06216 (-0.0)	0.028 (+11.0)	25.58 (+0.05)
0.12497	0.059	27.28	0.12486 (-0.1)	0.052 (+14.1)	27.33 (+0.05)
0.24988	0.118	29.85	0.24973 (-0.1)	0.104 (+14.2)	29.93 (+0.08)
0.49993	0.235	33.43	0.49960 (-0.1)	0.201 (+16.7)	33.53 (+0.10)
0.99995	0.453	38.19	0.99957 (-0.0)	0.396 (+14.4)	38.31 (+0.13)
1.99945	0.847	43.83	1.99856 (-0.0)	0.754 (+12.4)	43.91 (+0.09)

Average (Bike, Café, Woman)

option bpp	option s/Mpel	option PSNR	para bpp (+/- %)	para s/Mpel (speedup %)	para PSNR (+/- dB)
0.06227	0.035	22.75	0.06232 (+0.1)	0.027 (+29.2)	22.77 (+0.02)
0.12488	0.063	24.75	0.12485 (-0.0)	0.056 (+11.8)	24.76 (+0.01)
0.24987	0.130	27.44	0.24975 (-0.0)	0.110 (+17.5)	27.48 (+0.04)
0.49986	0.241	31.10	0.49967 (-0.0)	0.208 (+15.8)	31.18 (+0.08)
0.99979	0.469	35.92	0.99972 (-0.0)	0.405 (+15.8)	36.03 (+0.11)
1.99889	0.880	42.14	1.99839 (-0.0)	0.787 (+11.8)	42.23 (+0.09)

Table 17 *Comparison of CPU times (and PSNR) between the modified EBCOT coder with parallel options, "-Crestart"*

and sub-block causal contexts, and the "lazy" coding mode with the same parallel options. Sub-block and code-block dimensions are 8x8 and 64x64, respectively.

Aerial 2 (2048x2048+A45

par8 bpp	par8 s/Mpel	par8 PSNR	lazy bpp (+/-%)	lazy s/Mpel (speedup%)	lazy PSNR (+/-dB)
0.06222	0.023	24.58	0.06241 (+0.3)	0.022 (+6.3)	24.60 (+0.01)
0.12425	0.054	26.46	0.12493 (+0.5)	0.045 (+21.3)	26.48 (+0.02)
0.24976	0.104	28.52	0.24993 (+0.1)	0.092 (+12.9)	28.53 (+0.00)
0.49947	0.197	30.58	0.49985 (+0.1)	0.181 (+8.7)	30.58 (+0.00)
0.99937	0.349	33.20	0.99956 (+0.0)	0.319 (+9.5)	33.21 (+0.01)
1.99833	0.702	38.03	1.99888 (+0.0)	0.644 (+9.0)	38.07 (+0.04)

Bike (2048x2560)

par8 bpp	par8 s/Mpel	par8 PSNR	lazy bpp (+/-%)	lazy s/Mpel (speedup%)	lazy PSNR (+/-dB)
0.06242	0.028	23.72	0.06245 (+0.0)	0.027 (+1.6)	23.71 (-0.01)
0.12468	0.059	26.24	0.12490 (+0.2)	0.051 (+15.1)	26.20 (-0.04)
0.24989	0.124	29.48	0.24963 (-0.1)	0.107 (+16.8)	29.41 (-0.07)
0.49945	0.221	33.34	0.49977 (+0.1)	0.197 (+12.0)	33.23 (-0.11)
0.99999	0.419	37.92	0.99944 (-0.1)	0.369 (+13.5)	37.76 (-0.15)
1.99732	0.805	43.87	1.99999 (+0.1)	0.666 (+20.8)	43.72 (-0.14)

Café (2048x2560)

par8 bpp	par8 s/Mpel	par8 PSNR	lazy bpp (+/-%)	lazy s/Mpel (speedup%)	lazy PSNR (+/-dB)
0.06237	0.026	19.00	0.06236 (-0.0)	0.028 (-8.2)	19.00
0.12500	0.058	20.71	0.12493 (-0.1)	0.053 (+10.9)	20.71 (+0.00)
0.24964	0.103	23.04	0.24981 (+0.1)	0.103 (-0.1)	23.04
0.49997	0.203	26.67	0.49998 (+0.0)	0.188 (+8.2)	26.66 (-0.01)
0.99959	0.401	31.85	0.99946 (-0.0)	0.377 (+6.3)	31.80 (-0.05)
1.99928	0.802	38.90	1.99991 (+0.0)	0.693 (+15.6)	38.83 (-0.07)

Woman (2048x2560)

par8 bpp	par8 s/Mpel	par8 PSNR	lazy bpp (+/-%)	lazy s/Mpel (speedup%)	lazy PSNR (+/-dB)
0.06216	0.028	25.58	0.06246 (+0.5)	0.024 (+15.7)	25.58
0.12486	0.052	27.33	0.12469 (-0.1)	0.052 (-0.7)	27.32 (-0.01)
0.24973	0.104	29.93	0.24989 (+0.1)	0.099 (+4.3)	29.92 (-0.01)
0.49960	0.201	33.53	0.49954 (-0.0)	0.186 (+8.0)	33.52 (-0.01)
0.99957	0.396	38.31	0.99999 (+0.0)	0.361 (+9.8)	38.28 (-0.03)
1.99856	0.754	43.91	1.99928 (+0.0)	0.653 (+15.4)	43.91 (-0.01)

Average (Bike, Café, Woman)

par8 bpp	par8 s/Mpel	par8 PSNR	lazy bpp (+/-%)	lazy s/Mpel (speedup%)	lazy PSNR (+/-dB)
0.06232	0.027	22.77	0.06243 (+0.2)	0.026 (+2.4)	22.77 (-0.00)
0.12485	0.056	24.76	0.12484 (-0.0)	0.052 (+8.4)	24.74 (-0.02)
0.24975	0.110	27.48	0.24978 (+0.0)	0.103 (+7.1)	27.45 (-0.03)
0.49967	0.208	31.18	0.49976 (+0.0)	0.190 (+9.4)	31.14 (-0.05)
0.99972	0.405	36.03	0.99963 (-0.0)	0.369 (+9.9)	35.95 (-0.08)
1.99839	0.787	42.23	1.99973 (+0.1)	0.671 (+17.3)	42.15 (-0.07)

IV RESULTS REPORTED IN SPREADSHEET

The spreadsheets supplied by Alan Chien will be filled out for the following modes to provide a larger set of test results under a subset of the conditions for which partial results are reported in this document:

- VM4.1 EBCOT coder
- VM4.1 "option" coder with "-Ccausal"
- Combined modifications described in Sections II-1 and II-2.
- Parallel option ("-Crestart" and sub-block causal contexts, 8x8 sub-blocks)

V CONCLUSIONS AND RECOMMENDATIONS

From the material presented in this document, the following conclusions may be drawn.

- The modifications to the original EBCOT algorithm described in Section II-1 have negligible effect on compression performance, while simplifying the implementation of the algorithm. We recommend that they be adopted forthwith.

- The alternative sub-block significance coder described in Section II-2 leads to very minor improvements in compression performance. We recommend that it be adopted on the basis that it avoids some of the controversial issues currently surrounding the use of quad-tree techniques (hopefully, this situation will change) and also because, unlike quad-tree coding, its coding efficiency does not rely upon the fact that the number of sub-blocks in the width and height of the code-block are both approximately equal to 2^n for some integer, n. Thus, for elongated blocks we expect the quad-tree technique to suffer, although we have not had time to investigate this.

- The options described in Section II-3 introduce the possibility of exploiting "microscopic parallelism" in hardware implementations, and perhaps also reducing the external memory consumption in some applications. Because these options hurt performance somewhat and are not of interest to all applications, we recommend that they be adopted as options, rather than as mandatory. These options accomplish the same objectives as the existing "option" coder in VM4 while substantially reducing the number of symbols which must be coded, increasing execution speed of software implementations, and yielding somewhat higher compression performance. In fact, the EBCOT coder in VM4, the modifications and options proposed here and the "option" coder in VM4 are all very close relatives of one another. Based on the experimental evidence provided here, there does not appear to be any need to preserve multiple distinct entropy coders in the VM; the "option" coder should be removed and the modifications suggested in this document, many of which utilize similar or identical principles to the "option" coder, should be included instead.

- The "lazy" coding mode described in Section **Error! Reference source not found.** introduces the possibility of very substantial reductions in the number of symbols which must be coded at higher bit-rates and, especially, in lossless compression applications. Performance is degraded relatively little at moderate bit-rates and lossless compression is actually improved. For the moment, we recommend the adoption of this variation as an option.
- Based on the results presented in this report, there is reason to suggest that we might standardize on a sub-block size of 8x8, rather than the current default sub-block size of 16x16. Performance is reduced very little and software execution time and symbol count are somewhat reduced by the use of 8x8 sub-blocks. Moreover, 8x8 sub-blocks almost certainly lend themselves to more compact, less expensive hardware implementations. The hardware implementation suggestions in WG1N1020R for exploiting the compactness of the sub-block scan would obviously benefit from the use of smaller sub-blocks. Finally, smaller sub-blocks enhance the opportunities for parallelism.